2014

Improving the Selection of Surrogates During the Cold-Start Phase of a Cyber Foraging Application to Increase Application Performance

Brian Kowalczk
Nova Southeastern University, bkowalczk@hotmail.com

Improving the Selection of Surrogates During the Cold-Start Phase of a Cyber Foraging Application to Increase Application Performance

by

Brian A. Kowalczk

A dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in
Computer Information Systems

Graduate School of Computer and Information Sciences
Nova Southeastern University
2014

We hereby certify that this dissertation, submitted by Brian Kowalczk, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

_____ _____

Gregory E. Simco, Ph.D. Date
Chairperson of Dissertation Committee

_____ _____

Francisco J. Mitropoulos, Ph.D. Date
Dissertation Committee Member

_____ _____

Sumitra Mukherjee, Ph.D Date
Dissertation Committee Member

Approved:

_____ _____

Eric S. Ackerman, Ph.D. Date
Dean, Graduate School of Computer and Information Sciences

Graduate School of Computer and Information Sciences
Nova Southeastern University

2014

An Abstract of a Dissertation Submitted to Nova Southeastern University
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Improving the Selection of Surrogates During the Cold-Start Phase of a Cyber Foraging Application to Increase Application Performance

by
Brian A. Kowalczk
July 2014

Mobile devices are generally less powerful and more resource constrained than their desktop counterparts are, yet many of the applications that are of the most value to users of mobile devices are resource intensive and difficult to support on a mobile device. Applications such as games, video playback, image processing, voice recognition, and facial recognition are resource intensive and often exceed the limits of mobile devices.

Cyber foraging is an approach that allows a mobile device to discover and utilize surrogate devices present in the local environment to augment the capabilities of the mobile device. Cyber foraging has been shown to be beneficial in augmenting the capabilities of mobile devices to conserve power, increase performance, and increase the fidelity of applications.

The cyber foraging scheduler determines what operation to execute remotely and what surrogate to use to execute the operation. Virtually all cyber foraging schedulers in use today utilize historical data in the scheduling algorithm. If historical data about a surrogate is unavailable, execution history must be generated before the scheduler's algorithm can utilize the surrogate. The period between the arrival time of a surrogate and when historical data become available is called the cold-start state. The cold-start state delays the utilization of potentially beneficial surrogates and can degrade system performance.

The major contribution of this research was the extension of a historical-based prediction algorithm into a low-overhead estimation-enhanced algorithm that eliminated the cold-start state. This new algorithm performed better than the historical and random scheduling algorithms in every operational scenario.

The four operational scenarios simulated typical use-cases for a mobile device. The scenarios simulated an unconnected environment, an environment where every surrogate was available, an environment where all surrogates were initially unavailable and surrogates joined the system slowly over time, and an environment where surrogates randomly and quickly joined and departed the system.

One future research possibility is to extend the heuristic to include storage system I/O performance. Additional extensions include accounting for architectural differences between CPUs and the utilization of Bayesian estimates to provide metrics based upon performance specifications rather than direct observations.

Acknowledgements

First, I would like to thank my family for their endless support, encouragement, patience, and understanding throughout the dissertation process. To my children, Erik, Andrew, and Abigail, for their support and patience while I worked on homework when they did not have any schoolwork.

Next, I would like to thank Dr. Simco for his encouragement and guidance as chair of my dissertation committee. I would also like to extend my thanks to my dissertation committee members, Dr. Mitropoulos and Dr. Mukherjee for their guidance and support.

Finally, I would like to extend my thanks and gratitude to everyone that provided encouragement, advice, and support throughout my doctoral work.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Mobile devices are less powerful, more constrained, and tend to continually lag behind desktop workstations in terms of memory capacity, storage capacity, processor power, network bandwidth, and battery lifetime (Satyanarayanan, 1996; Verbelen, Simoens, De Turck, & Dhoedt, 2011). At the same time, many of the most useful applications to a mobile user include games, video playback, video editing, audio processing, voice recognition, facial recognition, and image processing, tend to be resource intensive and difficult to support on a mobile device (Balan, Gergle, Satyanarayanan, & Herbsleb, 2007; Chun, Ihm, Maniatis, & Naik, 2010; Narayanan, Flinn, & Satyanarayanan, 2000).

Despite the fact that mobile devices are resource constrained and therefore less capable than stationary workstations, users expect the same capabilities from them as they do from their workstation counterparts (Liagouris, Athanasiou, Efentakis, Pfennigschmidt, Pfoser, Tsigka, & Voisard, 2011; Verbelen, Simoens, De Turck, & Dhoedt, 2012). To bridge this gap between a device's capabilities and user expectations, Balan, et al. (2002) proposed an approach to augment mobile devices, called cyber foraging.

This research achieved the goal of developing a cyber foraging scheduling algorithm that decreased a cyber foraging application's execution time by eliminating the cold-start state. The new scheduling algorithm combined a historical algorithm with an estimation-based heuristic. The new experimental algorithm performed better than the historical algorithm and random scheduling algorithms in every operational scenario.

The remainder of this section presents the three common goals of cyber foraging followed by the demonstrated benefits of cyber foraging and concludes with an introduction to the cold-start problem and a discussion of the associated costs of the cold-start problem.

Cyber foraging systems attempt to balance the high expectations users place upon their mobile devices against the constraints of the device itself. The cyber foraging methodology selects and offloads code from a mobile device to a surrogate device for remote execution in an effort to increase the application's performance. Cyber foraging attempts to increase an applications performance by maximizing one or more of the following goals: decreasing the overall execution time, conserving power, or by increasing the fidelity of the response beyond what is otherwise possible with the current device (Balan, Flinn, Satyanarayanan, Sinnamohideen, & Yang, 2002; Satyanarayanan, 2001).

When the overall goal of a cyber foraging system was focused on reducing the overall execution time, the question that needed to be answered was whether the cost (in time) to execute a task locally was greater than the cost of remotely executing the same task. The basic formula for this decision took the form of: $C_L > (C_R + C_C)$, where C_L was the cost for a local task execution, C_R was the cost for remote execution, and C_C was the round-trip communication cost (Sharifi, Kafaie, & Kashefi, 2011). Anytime ($C_R + C_C$) was less than C_L, then the task was a candidate for remote execution.

The goal of reducing the energy consumption of an operation could be achieved if the energy consumed by executing a method remotely was less than the energy cost in performing the same operation locally, including the energy expended communicating

with the surrogate performing the operation (Verbelen, et al., 2012). Using the same formula presented above: $C_L > (C_R + C_C)$, where C_C was be expanded to $C_C = C_{TX} + C_{RX}$, where C_{TX} was the communication cost of invoking the remote execution, including the transmission of parameter data, and C_{RX} was the cost to receiving the results of the remote execution. If $(C_R + C_C)$ was less than C_L, then the task was a candidate for remote execution on the basis that it would conserve local battery power.

Fidelity is an application-specific notion that consists of one or more dimensions that include: size (in bytes), resolution, frame rate, bandwidth, and latency (Noble, Satyanarayanan, Narayanan, Tilton, Flinn, & Walker, 1997). Examples of fidelity in common use today include the resolution and frame rate of a streaming video and the size and resolution of a digital photograph. Since fidelity is an application specific concept, each application must provide hints about an application's fidelity dimensions to guide application developers in cyber foraging decision making (Narayanan, et al., 2000).

Several research projects have demonstrated the benefits of cyber foraging. The Spectra system demonstrated that a cyber foraging system could select the best remote execution plan the majority of the time (Flinn, Park, & Satyanarayanan, 2002). Odyssey, an early cyber foraging system, demonstrated that the battery life of a device could be extended by offloading code execution to surrogate computers. The offloading of code execution was shown to extend the battery life of a device by realizing an energy savings of up to 44% beyond what local hardware-based power management alone could deliver (Flinn & Satyanarayanan, 1999). Cuckoo, an offloading framework for the Android platform, demonstrated that it was possible to speed up computational tasks by a factor of 60 by offloading computationally intensive work to a more capable surrogate machine,

and, at the same time, reduce the energy consumption by a factor of 40 (R. Kemp, Palmer, Kielmann, & Bal, 2012). The AIOLOS system demonstrated that method offloading to a surrogate resulted in up to a 90% decrease in method execution time over local execution (Verbelen, et al., 2012).

The aforementioned systems were effective in part because beneficial offloading decisions were made by utilizing observed or historical performance data. These systems utilized performance metrics from prior executions and training to decide how to partition the task between local and remote execution (Flinn, et al., 2002; Flinn & Satyanarayanan, 1999; R. Kemp, et al., 2012; Narayanan, et al., 2000).

A common practice used to obtain performance metrics was to execute tasks on each remote system in order to obtain performance data. Kafaie, Kashefi, & Sharifi (2011) observed that in systems that utilized historical-based estimation, the system did not provide accurate estimates when there was a lack of observed performance data. Sharifi, et al. (2012) observed that a similar condition existed in historical-based estimation systems. When there was insufficient history to be utilized in estimation efforts, the cost estimates were also inaccurate. This condition was known as the cold-start state.

The effects of the cold-start state on offloading decisions can be illustrated by how the Odyssey system predicted future energy demand. The Odyssey system predicted future energy demand based upon direct observations and historical data. Odyssey's estimation methodology utilized an exponential smoothing function in the form of $P_{estimate} = \alpha(S_{current}) + (1-\alpha)*(S_{history})$,where α represented the weight of the power usage, $S_{current}$ represented the current observed sample, and $S_{history}$

represented the past demand estimation. The value for α was dynamically set to 10% of the remaining battery power. During the cold start state, before there was prior execution history or current execution observations, $S_{current}$ and $S_{history}$ were both zero, which yielded zero as the future demand prediction. This inaccuracy resulted in the system making an arbitrary and possibly detrimental decision based upon the faulty cost estimate. In Odyssey, the effect of the cold-start state was obscured by a smoothing function and the duration of the testing, which ranged in time from 20 minutes to 2.75 hours (Noble, et al., 1997).

The Spectra system partially addressed the lack of information during the cold-start state by utilizing default predictors, which provided a generic cost estimate whenever a current sample was not available (Flinn, et al., 2002). The default predictors in Spectra were historical-based and relied on logged execution data to generate a linear model of resource usage using linear regression. While this solution provided an approach to handle the case where current execution results were unavailable, the approach did not address the problem of when a new surrogate was encountered and there was a lack of both current and historical data. Additionally, this approach introduced the additional cost of training overhead. Essentially, the Spectra system suffered from the same drawback of faulty estimates as Odyssey, but incurred additional overhead in the form of training cost.

The Spectra system was tested in three common usage scenarios: speech-to-text translation, document formatting, and speech recognition. In the speech-to-text translation evaluation, the historical database was seeded by a training session that consisted of processing 15 phrases so that the system could start with baseline data. Prior

to the document formatting evaluation, Spectra processed 20 documents, which allowed the system to learn the performance metrics for the document formatting operation. Prior to the natural language selection test, Spectra was trained by translating 129 sentences before the actual test was initiated (Flinn, et al., 2002). The training avoided the cold-start problem, but imposed training cost in terms of effort and time. The overall cost of training the system was the sum of the individual task execution costs, but this simplistic calculation did not take into the account the cost of logging the individual operations nor did it incorporate the cost of the space required for storing the logs.

An example of training cost can be found in the Odyssey system. The Odyssey system added approximately 20 ms of overhead to each task invocation while offline training in Odyssey required approximately 10 seconds to read and process a log file (Narayanan, et al., 2000). According to Flinn et al. (2012), the need for a learning phase was a drawback of history-based approaches, but a necessary one as the accuracy of the predictions increased over time as more data was collected.

Using observations made by Narayanan et al. (2000), the case where a user of a cyber foraging system encountered a new environment where no device had ever been used before, a training session was required for each device before the devices could be utilized. Using the published training overhead times mentioned earlier, each new device would incur a 10 second training delay. In a dynamic environment, where devices joined and departed the environment spontaneously, it was impossible to know a-priori which devices would be available at any given moment. Delaying remote operation execution could potentially degrade the application's performance by missing a surrogate while it was available.

This report contains 5 chapters sequentially organized as follows. Chapter 1 provides background and introduces the research problem. Chapter 2 presents a review of the relevant literature and discusses gaps in the existing research. Chapter 3 presents the methodology used to develop and test the proposed scheduling algorithm. Chapter 4 reviews the results obtained by conducting the experiments. Chapter 5 discusses the implications of the results, and suggests recommendations for additional work.

Problem Statement

Cyber foraging systems that utilize historical performance metrics in remote execution decisions encounter a period during the initial start-up where there is insufficient historical data available to make accurate estimations. This problem, known as the cold-start state, is the period of time when historical-based estimation algorithms are inaccurate due to insufficient data to enable accurate estimations (Serral, Valderas, & Pelechano, 2011).

Kafaie, et al. (2011) stated that historical-based estimation algorithms that do not possess prior execution data for newly encountered surrogates were likely to be inaccurate. In a similar statement, Sharifi, et al. (2011) stated that one of the shortcomings of the historical-based estimation approach was that the algorithms required prior execution data, which was not available for newly encountered surrogates.

It is important that cyber foraging systems obtain and maintain timely and accurate information pertaining to the cost of both local and remote operation execution in order to make informed offloading decisions; otherwise, the system may not select the surrogate that provides the most benefit to the user (Flynn, 2012; Sharifi, et al., 2011).

According to Kristensen and Bouvin (2000), the delay imposed by the cold-start state prevented beneficial surrogates from being utilized until the system was able to make predictions. Because of this, historical-based algorithms may have delayed the utilization of a potentially beneficial surrogate while the surrogate was profiled. This delay may have resulted in continued degraded performance until a new and more beneficial surrogate was profiled and utilized. A scenario illustrating the potential cost associated with the cold-start follows.

To show the benefits of remote execution, Kemp et al. (2009) demonstrated that remote execution could both reduce the response time and improve the fidelity at which the application operates to a point beyond what the local device itself can perform. While the authors' system was in foraging mode, the execution time of facial recognition operations was reduced by a factor of up to 60 over local execution by offloading computationally intensive operations to surrogate machines. The ability to outsource the execution of computationally intensive tasks to surrogates not only decreased the execution time of the operations, it also potentially increased the fidelity of the operations. Due to memory and processor constraints of the mobile device, it was not possible to perform recognition operations upon high-resolution images with high accuracy settings on the local device.

In this case, cyber foraging provided the ability to offload the computation to more suitable surrogates, which augmented the local device to a point where such operations were possible (Kemp, Palmer, Kielmann, Seinstra, Drost, Maassen, & Bal, 2009). These benefits could not be realized if the system encountered a new surrogate and the surrogate was still in the cold-start state when an operation was executed. If the

system did not have enough information about the cost of utilizing the surrogate, another surrogate would be used (if one were available) or the operation would have been executed locally causing the application to run up to 60 times slower, or not at all.

The cost and duration of the cold-start state in the Odyssey system was demonstrated by how Odyssey predicted the resource demands of an application. Odyssey attempted to maximize the fidelity experienced by the user or to minimize the power consumed by the device by utilizing both a training process and a subsequent learning process. The training process utilized historical execution logs, if they were available, for use in the learning phase where they were loaded and used to generate predictors that guided the system in making remote execution decisions during the application's execution. If historical logs were unavailable, they were synthesized during an offline training phase where a series of random executions were made across the entire spectrum of possible requests. The resulting data was then fed into the training process for use by the system (Narayanan, et al., 2000). According to Narayanan, et al. (2000), the training process was performed offline and took approximately 10 seconds per device to complete. The offline training precluded new surrogates from dynamically joining the system; however, if new surrogates were able to join the system at runtime, they would have encountered an approximately 10 second training delay, assuming a training log was available for use. This delay extended the cold start state and prevented the system from realizing the performance benefits of a surrogate.

The historical-based task execution framework proposed by Huerta-Canepa and Lee (2008) reduced the execution time of an application by offloading code execution to surrogates in an effort to minimize the execution time of an application. Code was

offloaded to remote surrogates if it was estimated that the local resources would fall below a threshold that supported the required application performance. This was accomplished by a statistical sampling of local resources and incorporating prior application performance history, if available. The offloading decision was based upon the expectation of local resources being available within a 95% confidence interval of the target threshold. In order for the sampling to be statistically significant within the stated confidence interval, 96 samples were required to move beyond the cold-start state. The sample size was calculated as follows: $(Z^2\ p\ (\ 1 - p\ )\ ) / C^{2,}$, given $Z = 1.96$, $p = 0.5$, and $C = 0.1$, where Z was the confidence level, p was the standard deviation, and C was the margin of error (Huerta-Canepa & Lee, 2008). The drawback of this approach was the number of samples required to achieve the desired confidence level might have delayed offloading and exacerbated the problem by the continued execution of code on the local device when remote execution would have been beneficial.

Dissertation Goal

This research achieved the goal of increasing the performance of a cyber foraging application in terms of decreasing the application's execution time. This goal was achieved by the implementation of an enhanced scheduling algorithm that utilized a heuristic to estimate the execution cost of an operation on a device during the cold-start state. This estimation-based algorithm was utilized until the historical-based profiling algorithm acquired enough data to predict an operation's execution cost. The solution extended the linear regression-based algorithm utilized by the Odyssey system into the enhanced historical-based algorithm. This new algorithm utilized a heuristic based upon the static analysis of Java bytecode rather than historical execution logs to estimate the

cost of remote execution. This heuristic was utilized until the system obtained enough data for the prediction algorithm to be beyond the cold-start state.

All surrogates were considered to be in the cold-start state until they attained a prediction accuracy of 20% or less. This value was used based upon the success and accuracy of predictions in the Odyssey system, where the system achieved an error range of 10% to 24% (90^{th} percentile relative error) of the predicted CPU demand vs. the observed CPU usage (Narayanan, et al., 2000).

The remainder of this section presents the high-level approach of how the success of this research was measured. More details on the proposed algorithms are presented in Chapter 5 of this document.

Three experiments were conducted to measure the performance of the new scheduling algorithm proposed in this research. The first experiment measured the performance of the cyber foraging application with a historical-based prediction algorithm. The second experiment measured the performance of the cyber foraging application with the experimental algorithm. The third experiment measured the performance of the cyber foraging application with a blind offloading algorithm. Each experiment consisted of 4 scenarios, each of which targeted a specific operating condition. The differing operating conditions mimicked common use-case scenarios for mobile devices and included disconnected operation, use in an over saturated environment, use in a slowly churning environment, and use in a quickly churning environment.

Each scenario consisted of 3 image manipulation operations upon a full-size image and a thumbnail-sized version of the same image. The operations were repeated

fifty times for each image size. A complete overview of the testing plan and testing environment is presented in the performance evaluation section of the methodology chapter.

Relevance and Significance

This section supports both the problem and the goal of the research by first discussing the background of the current methodology leading to the problem, the lack of information and timing that manifests the problem, and the how solving the problem is beneficial.

The users of mobile devices are likely to possess and use multiple diverse devices simultaneously, which is in stark contrast to the mainframe era where one computer served multiple simultaneous users (Gu, Nahrstedt, Messer, Greenberg, & Milojicic, 2004). Amongst mobile devices, heterogeneity is commonplace with the hardware platform, operating system, physical characteristics, communication protocols, and overall device capabilities vary from device to device. Compounding the sheer number of possible device configurations is the fact that mobile devices are generally less powerful and more restricted than stationary hardware and this trend is unlikely to be solved by Moore's law alone (Narayanan & Satyanarayanan, 2003).

While reviewing options to address the disparity between platforms, Gu et al. (2004) observed that rewriting individual applications to make efficient use of a specific platform's resources would have been prohibitively expensive and time consuming. With the typical lifespan of a mobile device averaging less than 12 months, an approach was needed that allowed for applications to make efficient use of existing hardware with little or no source code modifications (Balan, et al., 2007).

Satyanarayanan (2001) proposed the use of cyber foraging to bridge this gap by partitioning code execution between local execution and remote execution in an effort to increase the performance of an application. By utilizing metrics obtained from the current execution environment, it was possible to determine if the remote execution would be beneficial to the application's performance. By remotely executing code on a surrogate device, an application's performance may have been increased by conserving the host machine's battery power, reducing the overall execution time of the operation, or increasing the fidelity of the operation (Balan, et al., 2002; Verbelen, et al., 2012).

Sharifi, et al. (2012) observed that the information required to make the decision to execute an operation locally or remotely was unavailable or incomplete during the cold-start state, rendering the offloading decision inaccurate. As a result, operation executions during the cold-start phase may not have yielded the desired performance. These suboptimal decisions may have also been distracting to the user and caused them to become impatient or frustrated with the application's performance (Flynn, 2012; Huerta-Canepa & Lee, 2008).

The Odyssey system presented by Narayanan et al. (2000) sidestepped the runtime cold-start problem by both defining the surrogates that would be present in the environment and by training the surrogates in advance. By identifying and training the surrogates a-priori, the system selected the most appropriate surrogate and APIs to utilize; however, it also restricted the movements of the mobile system to areas where the system was already trained (Kristensen & Bouvin, 2010). This approach effectively moved the cold-start problem from runtime to system deployment. This would be impractical in highly dynamic environments, such as vehicular ad-hoc networks, where

the topology of the network cannot be known in advance and nodes may only be available for as little as 10 seconds (Wang & Li, 2009).

According to Kristensen and Bouvin (2010), in a highly dynamic mobile environment, the chance that an operation has been previously executed on any of the currently available surrogates was low. This created an information gap between what was known about a surrogate and the execution history required to make informed decisions. On the other extreme, if there were a large number of surrogates available, this would have created a burden on the scheduler to both store and process the information for use in scheduling decisions. This overhead, in terms of both the storage space required for storing the information and the processing overhead incurred in managing and utilizing the data in scheduling decisions, must be properly managed; otherwise, it may have a negative effect on performance (Kristensen & Bouvin, 2010).

In an effort to mitigate the lack of data during the cold-start, Flinn et al. (2002) implemented default predictors that supplied a value when there was a lack of historical data available. The default predictors were implemented as linear models that expressed resource demand as a scalar data value. While this provided missing data during the cold-start state, it made two important assumptions when applied to resource demand and execution time: first, that resource demand was linear and second, that a given task would always have the same execution time. These assumptions were not true as resource supply was highly dynamic and the execution time of tasks was commonly a function of the input data (Kristensen & Bouvin, 2010).

Mobile devices are generally less powerful that stationary devices in terms of memory, storage space, CPU power, and battery power. This disparity cannot be solved

by scaling the hardware without seriously compromising the portability and battery lifetime of the device. The sharing of resources via cyber foraging has shown to be beneficial; however, the majority of current approaches used to determine if remote execution would be beneficial utilized some form of online or offline profiling. This profiling required the operation execution history for each device, which may not exist when new surrogates were discovered. The delay imposed between the time when new surrogate was discovered and when the surrogate became available for use may prevent a cyber foraging application from realizing increased performance by utilizing a more beneficial surrogate. Conversely, the effort required to profile surrogates that will not be beneficial may cost more than the overall savings.

Barriers and Issues

Developers of mobile applications are tasked with delivering software applications on relatively resource poor mobile devices upon which users place high-performance expectations (Sharifi, et al., 2011). To further exacerbate this situation, the release cycle of new hardware is measured in months rather than years and the pressure to develop and ship software with the new hardware is tremendous (Balan, et al., 2007).

A shorter development cycle itself is burdensome for developers and the addition of cyber foraging to the application requirements list further complicates the overall design (Balan, Satyanarayanan, Park, & Okoshi, 2003). In addition to traditional application development considerations, Balan, et al. (2003) observed that cyber foraging requirements force developers to consider other design goals, including resource monitoring, application partitioning, and remote execution that may run counter to traditional application development guidelines and increase overall development time.

This research avoided the aforementioned issue by separating the cyber foraging code from the application code by the use of aspect-oriented programming (AOP). Aspect oriented programming allowed for the clean separation of code into separate modules, which were woven together at runtime. This separation allowed for the cyber foraging code to be applied to method calls without the targeted method calls being modified directly to support cyber foraging. This eased the burden on the application developer because it was unnecessary to consider the cyber foraging requirements while developing the methods to support the functional requirements of the application.

Historical-based prediction algorithms that estimate the cost of remotely executing code benefit from hints supplied by the programmer. These hints, supplied in a file separate from the application, contain information that provides insight into factors that influence the cost of executing the code. Some of these metrics include algorithmic complexity, fidelity limitations, and resource utilization (Flynn, 2012). The added burden placed on application developers to hand-generate external files for use by cyber foraging systems makes it unlikely that the developers will be willing or able to adequately cover all of the possible combinations that the application will encounter (Chun, et al., 2010).

The system developed for this research avoided the issue of overburdening the software developer by requiring the developer to provide a single tactic file, which contained the signatures of the operations that were candidates for remote execution. No other analysis of the methods was necessary.

To ease the burden on application programmers, automated techniques to quantify the cost of code execution have been developed and implemented. CloneCloud,

developed by Chun et al. (2010) was one such example. CloneCloud utilized dynamic profiling to ascertain the cost of code execution for use in the scheduling of operations without programmer input. This assisted the programmer, but the use of dynamic profiling required that code be executed on each device that required profiling. This introduced the cold-start problem into the system in the form of a training period.

The use of automated techniques to ease the burden placed upon application development is enticing, but the predominate use of dynamic profiling techniques in cyber foraging systems introduces the cold-start problem, which can decrease an application's performance (Flynn, 2012). Further complicating matters is the fact that runtime profilers add overhead, thus negatively affect performance.

This research avoided the use of application profilers and other high-overhead techniques discussed earlier by utilizing the time in milliseconds it took to initialize the system. The initialization time was then used to calculate the speed rating for the device by utilizing the number of JVM instructions the initialization code executed. These steps required developer support to implement, but once the code was in place the metrics were dynamically calculated during system initialization.

Binder and Hulaas (2006) observed that applications profiled with the Java Virtual Machine Profiler Interface (JVMPI) experienced slowdowns ranging from a factor of 10 to a factor of 4000. The automatic profiling operations to obtain a cost estimate without running the code to obtain direct observations (thus avoiding the cold-start problem) suggested that a static analysis approach might be required.

The static profiling of Java applications to extract cost metrics using bytecode was complicated by Java's use of unstructured flow of control (the goto statement), stacks,

and virtual methods (Albert, Arenas, Genaim, Puebla, & Zanardini, 2007). The use of the unstructured *goto* statement hampered static analysis by increasing the number of edges in the flow analysis, thus increasing the size of the graph. Java's use of stacks to hold local variables limited the visibility of variables making it difficult to utilize them in the analysis. Virtual method invocations make it impossible to determine statically which method would be invoked at run-time because the data type of the object referencing the method was unknown (Albert, et al., 2007). The use of bytecode rather than source code was advantageous because access to an application's source code could not be guaranteed.

Further complicating estimation efforts was the fact that the complexity of an operation was often a function of the size of the input parameters (Kristensen & Bouvin, 2010). This impaired the ability to estimate the cost of operations, especially if the cost was not a linear function of the input parameter(s). This problem was further compounded by the differences in architecture, notably CPU architecture. Kristensen (2010) observed that the architectural differences between the Intel CPU architecture and the PowerPC CPU architecture generated a variance in the task weighting that was up to three times higher than the weight of the same function on an Intel processor.

This research avoided the application profiling overhead by generating control flow graphs (CFG) of methods in order to calculate the average number of JVM instructions contained within the method. This static analysis was performed once for library functions upon their addition to the code repository and upon the application itself at run-time when the cyber foraging system was initialized. This approach avoided the

overhead of profiling tools and the use of CFGs enabled Java's unstructured bytecode to be traversed using a graph traversal.

Assumptions, Limitations and Delimitations

The closed nature of the network used in this research and the sequential nature of the experimental scenarios allowed for the assumption that the communication latency between nodes was constant. This allowed the communications latency to be factored out of the performance results. Any variations in the network latency between individual nodes may have skewed the results if the communication latency varied significantly.

Due to resource constraints, the surrogate pool was limited to 5 surrogate machines. These machines are diverse in architecture, CPU speed, available memory, and storage. The decision to limit the number of machines may not stress the scheduling algorithms as much as they may be in highly populated areas. This may have allowed algorithmic issues due to scaling to go unnoticed.

Definition of Terms

Term	Definition
Advice	The code defined to run when the pointcut identifies a join point.
Android	Android is a popular mobile operating system developed by Google.
Aspect Oriented Programming	A programming method that is used to separate distinct tasks in a program that would otherwise be combined (tangled) together for convenience rather than functionality.
Cold-Start Problem	The condition created when there is insufficient information available to make decisions based upon inferences drawn from the data.

Cold-Start State	The period in time when a system is susceptible to the cold-start problem.
Control Flow Graph	A graph-based representation of the possible execution path(s) a function may take during execution.
Cyber Foraging	A method of extending a device's capabilities by utilizing services and resources provided by devices in the nearby environment.
Estimation	Calculation that may be determined based upon a heuristic rather than an exhaustive calculation.
Execution Time	The amount of time it takes to execute a function from the time the function is called to when the function returns the results.
Historical-Based Prediction	A calculation that utilizes past known values for solving an problem to establish a relationship with future values often used with linear regression.
Heuristic	Method to quickly arrive at an answer; however, the answer may not be optimal. Heuristics generally are faster than the polynomial time required to solve the same problem for an optimal solution.
Joinpoint	Defines the position in an executing program or within a static program.
Linear Regression	A method used to model a relationship between one or more variables in a series of data points.
NP-Complete	A set of problems that can be solved in polynomial time.
Pointcut	An expression that defines a pattern to be matched against a program's join points.
Polynomial Time	The time required to solve a problem expressed as a polynomial.
Scheduling	The process of determining where to execute a job so that it maximizes the overall goal of the system.

Surrogate	An untrusted and unmanaged device that provides services to nearby clients.
Remote Execution	See Remote Procedure Call
Remote Procedure Call	A method of executing code on another device transparent of the network providing the illusion that the code were being executed locally.
Fidelity	The degree to which the quality delivered by a service compares to the quality of the original source.
Partition	The code selected to be offloaded to a surrogate for remote execution.
Partitioning	The process of selecting code that may be offloaded to a surrogate for remote execution in a cyber foraging system.

Summary

Mobile devices due to their size, weight, and power constraints typically lag behind stationary desktop workstations where processing power, memory, and storage capacity are concerned. The cyber foraging paradigm enables mobile devices to perform beyond their means by offloading code for remote execution. By remotely executing code, an application can conserve memory and battery power by allowing surrogate machines to expend the resources rather than requiring the mobile device itself to expend the precious resources. The remote execution of code may also allow for the overall execution time of the process to be shortened or the fidelity of the result to be increased due to the utilization of high-performance computers rather than the resource poor mobile device.

A barrier to making offloading decisions in a cyber foraging system centered on obtaining enough information to make informed remote execution decisions. Given ample time and processing power, an execution scheduler could enumerate all available surrogates to determine the optimum surrogate to utilize in a given situation; however, as the number of surrogates increased, the time required to make such a determination would also increase and may become greater than what the end-user would be willing to accept. The price would also be increased in terms of both the processing power and the battery power that would be expended to make the decision. This could increase the cost of making the offloading decision beyond what would be saved by remotely executing the operation. This scenario may also be compounded by the cold-start problem. The cold-start problem could delay the availability of a newly arrived surrogate because the

system does not have enough information available to schedule the newly arrived surrogate.

The achieved goal of this research was to investigate if metrics obtained from the run-time profiling of a Java program could be utilized by an estimation algorithm to help a cyber foraging system make beneficial offloading decisions during the cold-start state thereby increasing an application's performance. The utilization of run-time metrics from the applications themselves provided a heuristic that did not require a-priori training, design-time information from the developer, or training effort from the end-user in order for the system to make informed offloading decisions that benefited the end-user.

The next chapter presents a review of the relevant literature and includes the cold-start problem, a review of the methods utilized to address the cold-start problem, including the use of default values or actions, historical-based algorithms, and heuristic-based approaches. The strengths and weaknesses of existing work are identified and gaps in the current approaches are identified and discussed.

Chapter 2

Review of the Literature

Introduction

This research achieved the goal of increasing the performance of a cyber foraging application during the cold-start state by augmenting a history-based prediction algorithm with an estimation algorithm to avoid the cold-start state. The overall goal of utilizing cyber foraging in this research was to augment the capabilities of a resource constrained mobile device by utilizing resources present in the local environment, thereby enabling the constrained device to exceed its capabilities to better meet the needs of the user (Balan, et al., 2002). Past cyber foraging systems attempted to increase performance by minimizing an application's execution time, minimizing energy consumption, or maximizing the fidelity of the content (Balan, et al., 2003; Cuervo, Balasubramanian, Cho, Wolman, Saroiu, Chandra, & Bahl, 2010; Kristensen & Bouvin, 2010; Verbelen, et al., 2012).

The scope of this literature review includes discussions on cyber foraging scheduling algorithms, which include scheduling algorithms from the related domains of grid computing, cloud computing, and peer-to-peer systems. This section begins with an overview of the cold-start state in cyber foraging systems and continues with discussions on scheduling algorithms that utilize default values or actions, historical-based prediction, and heuristics to make scheduling decisions.

The Cold-Start Problem

The cold-start problem, first discussed in recommendation systems, referred to a recommendation request for an item when recommendation data did not exist for the

item. This situation was often caused by the newness of the item and occurred when users did not have ample time to obtain, use, and comment on an item.

This scenario is common in websites that offer users' ratings as part of a search option. The adverse effects of the cold-start problem in a retail scenario may cause consumers to not see new items if the search query contains a ranking attribute. This is a result of the system's inability to provide a recommendation because there is no basis to form a recommendation (Schein, Popescul, Ungar, & Pennock, 2002).

Default-Based Algorithms

To avoid the cold-start problem in a pervasive system, Serral et al. (2011) approached the problem by seeding a user preference dataset with the default actions to be used when a user preference was unavailable for a condition. By requiring the system developer to provide default actions for each possible scenario that could be encountered, the system avoided the cold-start problem by performing a default action until the system obtained enough data to learn a user's preference (Serral, et al., 2011). This approach effectively addressed the cold-start problem at the user-level, but this approach had two consequences. First, it required the system developer to do additional work by providing default actions for each scenario and second, it pushed the cold-start problem from the user-layer into the system layer.

By utilizing default actions at the user-layer, the cold-start problem was effectively pushed into the cyber foraging level where it was reasonable to assume that if the system did not have enough information to make a recommendation to the user, it did not have enough information to make remote execution decisions on behalf of the user. The cold-start problem manifested itself in a cyber foraging system by the presence of

one or more surrogates in the environment that the system dad never interacted with before. This situation leads to the inability of the system to utilize the unknown surrogates when making scheduling decisions because of a lack of information about the surrogate. Without data about the surrogate, the system did not have the information required to determine if utilizing the new surrogate would be more or less beneficial than utilizing one of the known surrogates.

Narayanan, et al. (2000) implemented a closed-system approach in the Odyssey system to avoid the cold-start state and constrained the system to a few known surrogates. The closed system approach used in Odyssey required that each surrogate be profiled in advance of joining the system. This advance profiling guaranteed that performance data about each surrogate would be available for use in scheduling decisions; however, the closed system approach has some disadvantages. The closed system approach is more suited to an individual's home or workplace where mobility is limited rather than in highly mobile environment, such as a bus station or an airport terminal, where ad-hoc surrogate encounters are likely.

The Spectra system, the successor to the Odyssey system, utilized default models to avoid the cold-start state in the situation where historical data were unavailable to predict resource demand (Flinn, et al., 2002). In Spectra, resource monitors were used to share resource levels between cyber foraging clients and servers to model the resource demand for use in offloading decisions. In the absence of data, Spectra used default resource demand models that were based upon linear models of resource consumption. These model supply predictions for unknown values based upon execution history and extrapolation. If a prediction was requested and the system was unable to find a suitable

model in the execution history, the system provided a generic estimate derived using linear regression. These demand models were similar to the default actions utilized by Serral et al. (2011), and shared the same weakness in terms of increased developer workload, because it required the developer to provide default monitors and models for each resource. Another concern with the use of default models was the appropriateness of the model across heterogeneous architectures.

Balan et al. (2002) proposed using a brute force approach to surrogate utilization. The proposed method would have avoided the cold-start problem by utilizing every surrogate present in the environment and taking the first response. Because every available surrogate would be utilized regardless if historical execution data were available, this approach would typically yield beneficial performance. This approach would also have avoided the uncertainty that accompanied predictions and was immune to the cold-start problem; however, the brute force approach has a serious drawback: the approach does not scale well as the number of surrogates increases. As the number of surrogates increases, the communication, memory, and processing costs also increase due to the increased management load. This increasing cost could quickly outweigh the savings realized by offloading operations (Balan, et al., 2002).

Historical-Based Algorithms

The majority of the research efforts in cyber foraging surrogate selection has focused on the use of historical-based profiling techniques (Kafaie, Kashefi, & Sharifi, 2011). According to Kafaie, et al. (2011), the bulk of prior cyber foraging research has utilized online profiling, which requires the use of historical datasets in the prediction of the execution time of operations on remote surrogates. The utilization of historical-based

algorithms to make predictions was enticing because the predictions generally increase in accuracy over time as more data was accumulated (Gurun, Krintz, & Wolski, 2004). However, Flynn (2012) noted that the downside of using historical-based algorithms to make predictions was the cold-start problem. The algorithms required a training period (the cold-start problem) in order to obtain sufficient data for use in generating predictions (Flynn, 2012). This delay may have caused opportunities to use beneficial surrogates to be missed due to a lack of data.

To quantify this delay, the profiling process in the Odyssey system will be used as an example. Profiling a surrogate in the Odyssey system was performed offline and took approximately 10 seconds per surrogate. This assumed that a historical dataset was available. If a dataset was available, this file was provided as input to the profiler. However, if a historical dataset did not exist, it was generated by a training session. This training session required that a surrogate repeatedly execute the required operation(s), often with varying input, to generate a historical dataset for use in profiling (Narayanan, et al., 2000). The training and profiling of surrogates had the potential of introducing a substantial delay between when a surrogate was first encountered and when it became available for use. To avoid the training penalty, Huerta-Camepa and Lee (2008) proposed incorporating the execution history from other surrogate devices during the integration of new surrogates into the system.

When a device travels to a new environment, there is a high degree of probability that it will encounter new devices and be requested to perform operations that the device has never performed before (Kristensen & Bouvin, 2010). This situation is at the heart of the cold-start problem. By importing the execution logs of other devices, a surrogate

could minimize the time spent in the cold-start state and be available for use faster (Huerta-Canepa & Lee, 2008; Narayanan, et al., 2000). There are several unsolved challenges associated with this approach. First, conversions would be required to account for the performance differences between heterogeneous architectures, including differences introduced by CPU architecture and hardware speed. Second, performance metrics may be platform dependent would need to be converted from one platform to another to ensure that a reasonable comparison is made (Narayanan, et al., 2000). Kristensen and Bouvin (2010) observed that the differences in platforms, including CPU architecture, compiler optimizations, and hardware architecture all contribute to the difficulty of finding a measure that can classify the power of heterogeneous machines. Such a classification would make it possible to group heterogeneous machines according to their respective power or throughput ratings.

Heuristic-Based Algorithms

According to Kafaie, et al. (2011), little work in cyber foraging surrogate selection has focused on utilizing approaches other than historical-based profiling. One reason for this may be due to the overall accuracy that these approaches offer over time (Gurun, et al., 2004). Although the delay imposed by profiling has been previously discussed, approaching the job of scheduling remote execution in a cyber foraging system from the perspective of grid computing provides a new perspective on the need to complete the scheduling task quickly.

The task of remote execution scheduling performed in a cyber foraging system can be viewed as a dynamic grid where the grid is comprised of surrogate devices. Job scheduling in a grid environment is an NP-complete problem that must be solved in a

relatively short period of time (Pooranian, Shojafar, Abawajy, & Singhal, 2013). Grid computing scheduling algorithms tend to favor optimizing makespan to reduce the overall execution time of a job stream, which is similar to the goal of reducing an application's execution time in this research. According to Pooranian et al. (2013), since job scheduling is a NP-complete problem that must be solved in a relatively short period of time, the use of deterministic algorithms is not ideal. Even though a deterministic algorithm would eventually yield the correct answer, for a large number of nodes, the algorithm may not arrive at the solution in a reasonable amount of time. Solving this type of time-sensitive problem favors heuristic algorithms over deterministic algorithms.

In an effort to avoid profiling and the need for historical datasets, the adaptable offloading inference engine (OLLIE) dynamically offloads classes to surrogate devices in an effort to reduce the memory consumption of a mobile device (Gu, Nahrstedt, Messer, Greenberg, & Milojicic, 2003). OLLIE utilizes developer supplied class annotations, a fuzzy control inference engine, and developer supplied rules to control adaptation decisions that dynamically partition the executing application at runtime into objects that may be offloaded and accessed remotely via remote method invocation. The fuzzy inference engine utilized by OLLIE requires developer support to provide fuzzy logic rules to determine when to trigger offloading. The intriguing aspect of OLLIE from the perspective of this research is that no a-priori knowledge of the surrogates or execution history is required for the system to make beneficial offloading decisions. This is due in large part because the goal of conserving memory on the mobile devices can be realized by remotely instantiating an object on a surrogate machine given there is adequate

memory available on the surrogate. Adaptation is initiated by the single heuristic trigger of the available memory on a remote device to execute offloading.

Zhang, Kunjithapatham, Jeong, & Gibbs (2011) proposed an elastic application model that would automatically partition an application into individual weblets that could be dynamically and independently offloaded into the cloud to augment and conserve a mobile device's resources. In an effort to determine the optimal balance between the number of offloaded weblets and locally executing code, a Naïve Bayesian Learning algorithm was utilized to keep the offloading balanced between the cloud and the mobile device. This was achieved by using a cost-based approach. The cost of specific resources and performance attributes were utilized by a learning algorithm and balanced against local resource measurements, historical performance data, and user preferences to control the partitioning of the application (Zhang, Kunjithapatham, Jeong, & Gibbs, 2011). Although this system utilizes a probabilistic approach over a deterministic approach to obtain the cost estimate this approach, like the Odyssey system, also suffers from the cold-start problem due to the dependence upon historical data to train the system before it can make predictions.

Kafaie, Kasherfi, and Sharifi (2011) presented a cost-based approach to the cold-start problem by using the throughput of an operation executed on a specific device as a cost metric that could be utilized to make scheduling decisions. The cost metric, instructionPmSecond, was defined as the quotient of the number of elements that required processing and the time required to perform the operation (Kafaie, et al., 2011). Ideally, the value of instructionPmSection would be computed in an offline training session; however, if a new surrogate was encountered at runtime that did not have a value

for instructionPmSection, the system profiled the operation dynamically to obtain the cost metric. Although this approach suffered from the same scaling problem as the brute-force approach presented by Balan et al. (2002), it had two strengths. First, the system did not refuse to allow new surrogates to participate if it had not been profiled in advance. Second, the use of the metric (instructionPmSection) was preferable to the use of execution time itself. This was a step towards a device independent metric, which could be used to quantify the strength of the operation when executed on the surrogate.

Using a similar approach, Kristensen et al. (2010) utilized benchmarking to assign a strength rating to surrogates for use as a scheduling heuristic. This heuristic enabled the Scavenger system to make beneficial offloading decisions when there was a lack of historical information. Scavenger's scheduler utilized two profiles: a peer-centric profile and a task-centric profile. The peer-centric profiles utilized historical information about the run-time of past executions in a (peer, task) pairing, while the task-centric profiles contained the weight of the task as if it were executed on a surrogate with a strength rating of 1. This scaling of the task weight by the strength rating of the surrogate allowed Scavenger's scheduler to make judgments about the best surrogate to use when a peer profile was not available. The strength ratings of the surrogates were linear where a surrogate with a strength rating of 2 was twice as fast as a surrogate with a strength weighting of 1 (Kristensen, 2010).

The benchmarking approach utilized in the Scavenger system provided relatively sound guidance to the Scavenger's scheduler; however, it was not perfect in every situation. Kristensen (2009) observed that architectural differences between platforms did influence the weights of tasks by as much as three times in some instances, which

may have led to inaccuracies in surrogate selection. Additionally, requiring the use of an external benchmarking application to obtain the surrogate strength was essentially an offline training phase.

An alternate approach to quantifying the strength of a surrogate was to quantify the resource demand of an operation. Binder and Hulaas (2006), in an effort to provide a cross-platform CPU consumption metric, utilized bytecode instruction counting as a method for quantifying CPU consumption of a Java application. The authors', motivated by the high overhead of profiling and lack of portability of the JVM Profiler Interface, utilized bytecode rewriting to count the number of JVM instructions executed by each thread of execution in a Java application. This approach enabled Java applications to be profiled with moderate overhead ranging from 17% to 30% of the applications run-time (Binder & Hulaas, 2006). The ability to describe the CPU consumption of a Java bytecode in a platform neutral metric enabled the metric to be used directly without the need to perform conversions or weight the value to account for variations on device performance.

A platform neutral metric avoided the need for platform specific conversions to account for architectural differences when estimating costs in a heterogeneous environment; however, the fact that the cost of an operation was often a function of the size of the input parameters also influenced the estimation. In an effort to glean cost relations from Java bytecode, Albert et al. (2007) utilized a CFG to convert Java bytecode into a traversable graph structure. The resulting CFG was used as input into a static analysis process designed to infer the operational complexity of the Java bytecode based upon the input parameters and the variables utilized to control branching and looping

within the program. Although obtaining cost relations was an important component of determining the complexity of an operation, which in turn was required to determine the running time of the operation, the focus of this work was not to determine execution time, but rather to determine which surrogate would potentially provide the fastest execution time. A CFG was utilized to calculate the longest, shortest, and average path of execution through an operation. The average path cost was utilized as a heuristic that indicated the overall cost of the operation rather than determining the exact cost of the operation using a deterministic method.

Chapter 3

Methodology

Introduction

This research attained the goal of increasing an application's performance during the cold-start state by designing and implementing an enhanced historical-based prediction algorithm. This algorithm utilized estimation for surrogate selection during the cold-start state of a cyber foraging application until the historical-based prediction algorithm accumulated enough execution history to make predictions. To provide an environment where the new algorithm could be evaluated, a Java-based cyber foraging system, called jScavenger, was developed using the Python-based Scavenger system developed by Kristensen (2009) as a model.

This chapter is organized as follows. First, a high-level overview of the jScavenger system will be presented, followed by a detailed discussion of the individual jScavenger components (the foraging application server, the jScavenger Surrogate client, and the cyber foraging application). Next, a discussion on the approaches used for profiling the operations and devices will be presented followed by discussions on the testing environment, performance evaluation, data collection, data analysis, and data verification processes.

jScavenger Overview

The jScavenger system was a Java-based client/server system where cyber foraging applications executing on a mobile device, such as a tablet or smartphone, remotely executed code in an effort to decrease the overall execution time of an application. Surrogate devices, located in the local environment, connected to the

jScavenger foraging application server (foraging server) to offer computational resources to cyber foraging applications. If the foraging server determined that the operation about to be performed would potentially run faster on a surrogate device, then the operation would be offloaded to a surrogate. The high-level organization of the jScavenger system is shown in Figure 1.

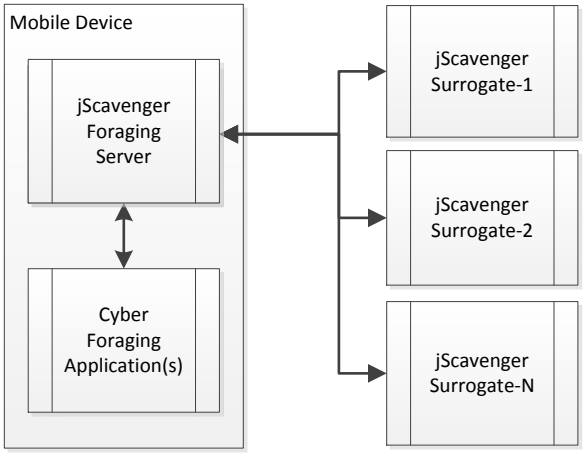

Figure 1 – A High-Level View of jScavenger

In Figure 1, the cyber foraging application depicted was an image manipulation application, which enabled the user to sharpen an image, adjust the contrast of an image, or convert the image to grayscale. This application was an Android application running on a smartphone, which allowed the user to select an image and the operation to perform upon the image. The application was also able to execute predefined scripts to automate the data collection phase of this research.

Image manipulation was chosen because high-resolution cameras are standard on most mobile devices and the ability to manipulate images before they are uploaded to photo albums or social media sites is desirable; however, applying these operations to high-resolution images is still demanding and resource intensive for mobile devices in terms of time and energy (Kristensen & Bouvin, 2010). According to Kristensen et al.

(2010), cyber foraging has been able to reduce the time it takes for a resource constrained device to perform a series of image operations on a high-resolution image from 150 seconds without cyber foraging to less than 20 seconds with cyber foraging.

Surrogates in the jScavenger system functioned as remote procedure call (RPC) engines that accepted RPC requests, performed the requested operations, and returned the results. Each surrogate connected directly to the foraging server and maintained a library of operations that were available for use. When a surrogate connected to a foraging server the list of available operations on the surrogate were compared with the current requirements of the cyber foraging application(s) currently connected to the foraging server. If a surrogate was missing an operation that was currently required, the discrepancy was resolved by the surrogate downloading missing operation(s) from the foraging server. All surrogates in this research were assumed to be able to perform any operation that the cyber foraging application requested and each surrogate would have the required operations downloaded in advance.

When a cyber foraging application attempted to perform an operation that was available on a surrogate, the foraging server intercepted the method execution request and determined if remote execution was beneficial. If remote execution was deemed to be potentially beneficial, the foraging server sent a RPC request to the selected surrogate along with the parameter data. The surrogate then performed the operation and returned the result to the foraging server. The foraging server then presented the result of the operation to the requesting application as if the operation was performed locally. Conversely, if remote execution was not deemed beneficial, then the application processed the operation locally.

The jScavenger Foraging Application Server

The jScavenger foraging server functioned as the cyber foraging resource manager for the mobile device by providing surrogate discovery and remote execution scheduling services to cyber foraging applications. The high-level architectural overview of jScavenger is shown in Figure 2.

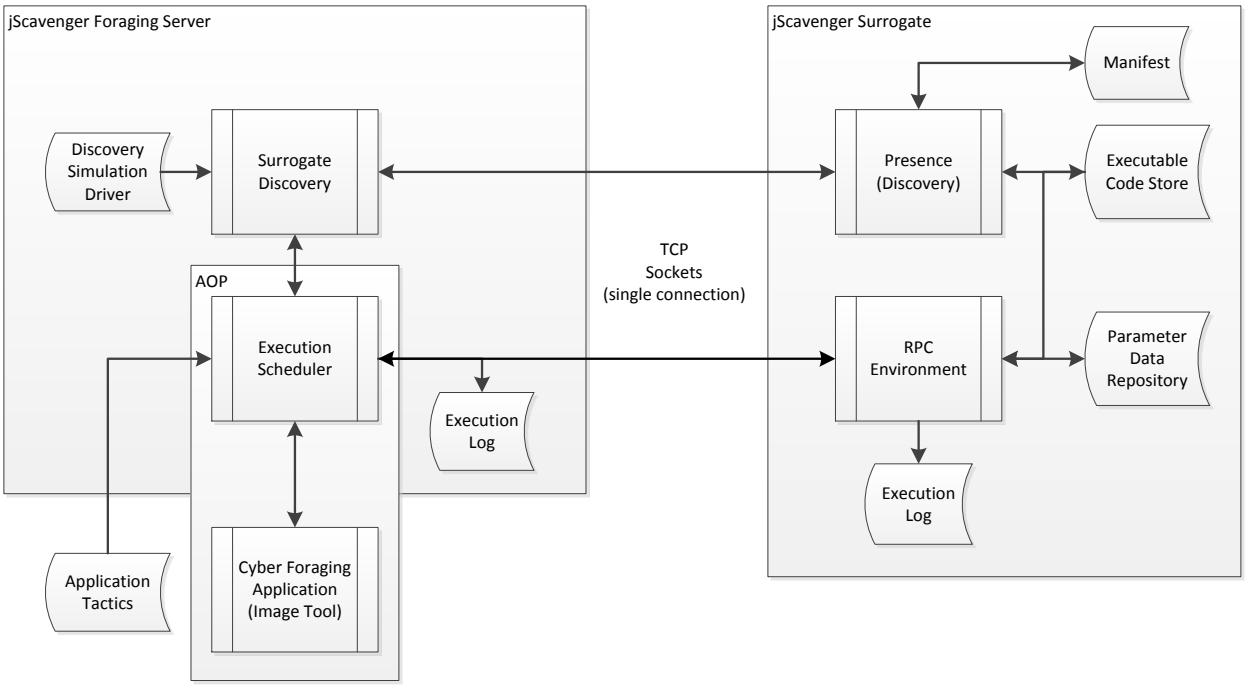

Figure 2 – The High-Level Architecture of jScavenger.

The Interface Between jScavenger and a Cyber Foraging Application

During the execution of a cyber foraging application, the foraging server intercepted the method calls of the cyber foraging application and determined if remote execution was beneficial. This interface was implemented using Aspect Oriented Programming. In order to intercept the calls from a cyber foraging application, an AOP aspect called jScavengerMonitor was created. This aspect contained a pointcut, which defined a predicate that was used to match the method calls to be intercepted. The basic form of the pointcut is shown below.

```
pointcut InterceptCalls():  call (* *(..))
```

This pointcut called the advice method InterceptCalls if the current join point matched the predicate defined in the pointcut. In this case, the predicate was a wildcard that intercepted all method calls. The advice InterceptCalls contains the logic to locally execute or remotely execute the call using method names from the application's tactic file and the list of available RPCs derived from the currently available surrogates. If the current method was available as a RPC and would potentially execute faster than local execution, then the request was submitted to the scheduler for execution; otherwise, the request was executed locally. The pseudocode of the InterceptCalls advice is shown in Figure 3.

```
Object around() : InterceptCalls()
{
       Object value = null

       if( Scheduler.isAvailable(rpcname) )
              value = Scheduler.execute(rpcname, params)
       else
              value = proceed(params)   // local execution

       return value
}
```

Figure 3 – Pseudocode for InterceptCalls Advice

In this research, cyber foraging was considered a non-functional requirement of the user application. The use of AOP enabled the cyber foraging related code to be cleanly separated from the source code of the cyber foraging application (Irwin, Kickzales, Lamping, Mendhekar, Maeda, Lopes, & Loingtier, 1997). This separation of concerns allowed cyber foraging services to be provided transparently to the application, thus avoiding the need to directly modify the application to support cyber foraging (Satyanarayanan, 1996).

The single requirement jScavenver imposed upon a cyber foraging application to utilize cyber foraging was that the application developer must have provided a tactics file that contained the names of methods that could be offloaded to a surrogate. If an application did not supply a tactics file, then the application will execute without the benefits of cyber foraging. The tactics file will be discussed in detail in the Cyber Foraging Application section.

The jScavenger Execution Scheduler

The jScavenger execution scheduler was responsible for determining the location where the current operation should be executed, remotely executing the operation (if applicable), and maintaining a log file that contains performance data about the system's operation. This section contains details on the scheduler, the RPC execution mechanism, and the execution log file.

The execution scheduler worked with the discovery subsystem to maintain a list of operations that may be remotely executed and a list of surrogates capable of performing the operations. At runtime, when a new surrogate connected to the foraging server, a manifest of the available operations was presented to the discovery subsystem, which in turn registered the surrogate with the execution scheduler. The execution scheduler then used the manifest and the surrogate's device name to maintain a list of operations that were currently available for remote execution. Each operation had the potential to be executed by none, one, or many surrogates. The relationship between the operations and surrogates is shown in Figure 4. For each operation, the associated surrogate list was maintained in order based upon the cost of performing the operation on the surrogate.

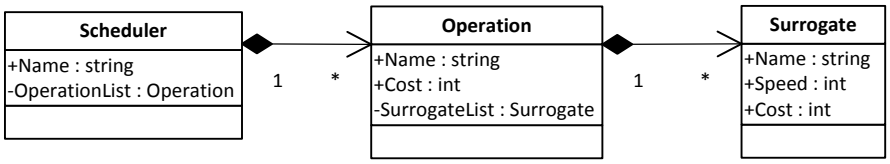

Figure 4 – Data Structure Mapping Operations to Surrogates

The execution scheduler in jScavenger determined whether to offload an operation to a surrogate or execute the operation locally using a cost-based metric. Kafaie, et al. (2011) developed a highly accurate solver that was able to successfully determine the most suitable surrogate to offload code execution using a cost-based model. Their cost-based solution defined cost functions for execution latency, computation time, communication time, and energy consumption based upon offline profiling. During experimentation, the authors' solution selected the best location to execute the task with a degree of accuracy that performed as well as blind offloading. Two of the drawbacks of this approach, which were addressed in this research, required the developer to annotate the complexity of each operation and the use of offline profiling to determine the speed of each device.

Based upon the research conducted by Kafaie, et al. (2011), the expression used to determine if an operation should be offloaded was: $IF\ C_L > (C_R + C_C)$, where C_L was the cost for a local task execution, C_R was the cost for remote execution, and C_C was the round-trip communication cost. If the estimated cost to perform the operation locally were greater than the sum of the estimated remote execution cost and the estimated round-trip communication cost, then the operation would be offloaded. The unit of measure for cost in this research was time, expressed in milliseconds. The method used to derive each cost varies by the scheduling algorithm that was used.

The jScavenger scheduling system contained three costing algorithms that were used to determine C_L and C_R: historical, experimental, and random. The foraging server's configuration file determined which scheduler was in use when the system was initialized.

The first algorithm, the historically-based algorithm, attempted to predict the C_R as the execution time of a given operation by performing an ordinary least squares linear regression. The calculation was performed over a historical dataset consisting of the operation execution times for a specific surrogate and the size of the image in pixels. This mirrors the approach utilized by Narayanan, et al. (2000) to predict the remote execution time on the Odyssey system. This research followed suit by utilizing simple linear regression to predict the remote execution time of an operation using the number of pixels contained within the image as input. Linear regression was implemented in jScavenger using the Apache Commons Math3 library and the observation data was stored in the scheduler for each connected surrogate.

This approach follows the Odyssey system's use of a linear regression-based algorithm that was able to predict the CPU demand for a given operation based upon the polygon count and resolution of the model to be processed. Using this approach, the Odyssey system achieved an error range of 10% to 24% (90^{th} percentile relative error) of the predicted CPU demand vs. the observed CPU usage (Narayanan, et al., 2000). It should be noted that these results were obtained after the Odyssey system was trained on each surrogate so the system had adequate data to make offloading decisions.

Like Odyssey, the jScavenger system's implementation of this algorithm suffered from the cold-start problem because the system was not trained on each individual

surrogates prior to executing the experiments. Although this algorithm was based upon the algorithm used by Odyssey; jScavenger, unlike Odyssey, accepted new surrogate connections at run-time. The requirement to allow new surrogates to join the system at run-time brought with it the cold-start problem due to the low probability that an operation was previously executed on the surrogate (Kristensen & Bouvin, 2010).

To address the cold-start problem, each new surrogate that did not have an execution history was profiled. This profiling was performed in the background before the surrogate was able to be utilized by the scheduler. The profiling was accomplished by requesting the surrogate to process two image files (one full-size and one thumbnail size) using the current operation. All surrogates were considered to be in the cold-start state until they attained a prediction error accuracy of 20% or less. Kafaie, et al. (2011) utilized a similar approach to gather performance information from an unknown surrogate. This was accomplished by transferring a small profiling program to the surrogate to gather performance metrics before the system could include the surrogate in scheduling calculations. Although this approach introduced additional overhead, experimental results showed that the overhead could be justified by enabling the scheduler to make better offloading decisions.

The second algorithm, the experimental algorithm (Figure 5) utilized a heuristic to estimate the most beneficial surrogate to utilize while the historically based algorithm was in the cold-start state. The heuristic estimated the remote execution cost as $C_R = O_C / D_S$, were O_C was the cost of the operation in terms of the average number of Java virtual machine instructions contained in the operation and D_S was the speed of the device in terms of the number of Java virtual machine instructions it demonstrated it could

execute per second. This heuristic was utilized until there was enough history accrued so that the historical-based algorithm would be sufficiently accurate. Using the error range from Odyssey as a guide, once the prediction error was below 20% of the observed value, the surrogate was considered to be beyond the cold-start state and the historical-based algorithm was utilized. The pseudocode for this algorithm is shown in Figure 5.

Kafaie, et al. (2011) utilized a similar metric to describe the performance of a device by performing offline profiling to obtain the number of data elements a device could perform in a second for a given task. The profiling utilized a developer supplied big-O expression for the time complexity behavior of the function and the element count of the input data as the workload. Although the term was named instructions per second, the value did not actually count machine instructions executed per second, but rather it represented the number of data elements that could be processed per second. This value was calculated by taking the number of elements in the data set divided by operation's execution time. This approach enabled the authors' to perform a brute force calculation over all surrogates to determine the best surrogate to utilize in a given situation (Kafaie, et al., 2011).

```
// Determine which device to use
device getDevice()
{
cost = localdevice.profile.getCost(operation)
host = localdevice
foreach device s in surrogates with operation
        cost_s = s.profile.getCost(operation)
        if( cost_s < cost )
                cost = cost_s
                host = s
        end-if
end-for
return host
}

// Determine the cost of the operation on a given device
double device.getCost(operation)
{
cost = infinity
if( device.ColdStart == true )
   cost = device.profile.getCost() * operation.profile.getCost()
else
   cost = device.history.getCost(operation)
return cost
}
```

Figure 5 - Pseudocode for the Estimation-Enhanced History-Based Algorithm

The use of a cost-based heuristic was utilized in the Scavenger system to enable the operations to be evaluated separately from the devices in what the authors' termed multidimensional profiles (Kristensen & Bouvin, 2010). Scavenger's use of multidimensional profiles enabled the characteristics of both the device and the operation to be reasoned about separately when making offloading decisions. For example, in the Scavenger system, a device with a strength level of 8 was considered to be twice as fast as a device with a strength level of 4. This type of direct comparison was not possible in a historical-based approach where the only data available was the time it took to execute the operation. By separating the characteristics of the device from the properties of the data to be acted upon, it became possible to estimate how a particular operation would perform on a specific device without actually performing the operation on the device

(Kristensen & Bouvin, 2010). Although the accuracy of historical-based approaches have proven themselves to be beneficial, they fall flat when presented with the cold-start problem (Kristensen & Bouvin, 2010).

One of the drawbacks of Scavenger's approach is that each device must be benchmarked offline to obtain the device's relative strength before it can participate in the system. The benchmarking suite NBench was used to provide the strength metric, which, according to the documentation, takes approximately 10 minutes to execute (Kristensen & Bouvin, 2010). The benchmarking requirement would delay devices that have not already been benchmarked by this specific software from participating in the system.

The jScavenger approach to profiling surrogates was as follows. First, the device was profiled to determine the estimated number of JVM instructions per second the device could perform. Second, the methods defined in the tactics file were profiled to determine the average number of JVM instructions for each operation. Once the device and operation profiles were obtained, the estimated execution time was calculated by dividing the operation profile cost by the device profile cost. The details on the methodologies used to profile the surrogate device and operations are discussed in the device profiling and operation profiling sections, respectively.

The third scheduling algorithm, the random algorithm, blindly selected a surrogate to use from the list of available surrogates. This algorithm utilized a random number generator that selects the surrogate to utilize for the current operation. This algorithm assumed that offloading was faster than local execution.

This research was conducted on a closed network to avoid unintended variations in network performance from influencing the overall performance of the testing environment. Because of this, the value of C_C was not estimated and was considered a constant value for each surrogate. The actual observed value of C_C was captured and evaluated to ensure that this assumption was valid.

The observed value of C_C was determined by capturing the total latency (denoted by C_T) as seen from the application server and subtracting the remote execution time as reported by the surrogate. The calculation used to determine C_C is illustrated in Figure 6.

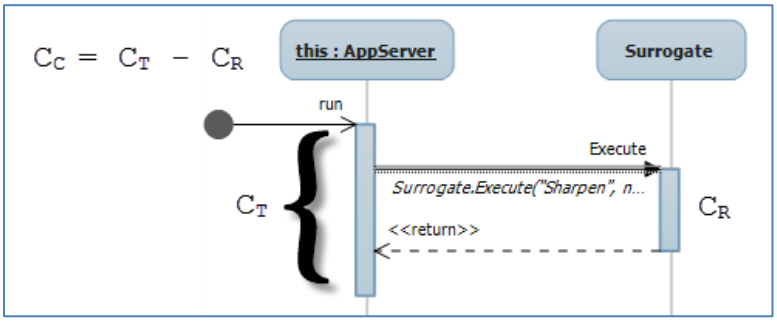

Figure 6 – Calculating Round-Trip Communication Cost

Two image sizes were utilized in this research, which introduced variability in the transfer rates used over the TCP connections. The TCP communication protocol utilizes a sliding window, which can dynamically change the number of active packets allowed to be outstanding at a given time to increase throughput. Because of this, the transmission rate for larger files was higher than the rate for smaller files (Kafaie, et al., 2011).

Remote Execution

When the scheduler decided to offload an operation to a surrogate, it created an RPC request formatted as a XML document and sent the RPC request via a synchronous TCP socket to the surrogate for execution. The RPC was then decoded, executed, and a response was returned. A detailed discussion about how the remote execution was

performed on the surrogate is provided in the jScavenger surrogate section. An example of an RPC request/response pair is shown in Figure 7 and Figure 8, respectively.

```
<?xml version="1.0" encoding="utf-8"?>
<request>
  <requestID>421</requestID>
  <requestType>RPC</requestType>
  <operationName>ImageLib.Contrast</operationName>
  <parameters>
    <parameter>
      <parameterName>RETURN_VALUE</parameterName>
      <parameterDirection>OUT</parameterDirection>
      <parameterDataType>byte[]</parameterDataType>
      <parameterValue>X62IBNhchbxBwbGhVwc==</parameterValue>
    </parameter>
    <parameter>
      <parameterName>IMAGE</parameterName>
      <parameterDirection>IN</parameterDirection>
      <parameterDataType>byte[]</parameterDataType>
      <parameterValue>W35IFNhchbCBwbGVhcw==</parameterValue>
    </parameter>
  </parameters>
</request>
```

Figure 7 – XML RPC Request

The RPC Request XML document contained all the information required by the surrogate to perform the RPC. The important elements of the request document were the operationName and parameterValue elements. The operationName parameter passed the class name and the method name to be remotely executed. The parameterValues parameter contained binary data (byte and byte[]) in base64 to avoid the possibility of introducing invalid characters into the XML document.

```
<?xml version="1.0" encoding="utf-8"?>
<response>
  <responseID>421</responseID>
  <operationName>ImageLib.Contrast</operationName>
  <executionTime>575000000</executionTime>
  <errorCode>0</errorCode>
  <errorDescription></errorDescription>
  <parameters>
    <parameter>
      <parameterName>RETURN_VALUE</parameterName>
      <parameterDirection>OUT</parameterDirection>
      <parameterDataType>byte[]</parameterDataType>
      <parameterValue>X62IBNhchbxBwbGhVwc==</parameterValue>
    </parameter>
  </parameters>
```

Figure 8 – XML RPC Response

The XML response reported upon the success or failure of the RPC. The response document contained an errorCode element, which contained the value of zero upon success, and a nonzero number upon failure. If the RPC failed, the errorDescription contained detailed information about the exception. The remote execution time (CR, in milliseconds) was returned in the executionTime parameter. If the operation contained output parameters or a return value, they were passed in the parameters array.

The Execution Log File

The foraging server maintained a tilde delimited execution log, which enabled the performance of the system to be analyzed. The log file was called "server.log" by default and contained three record types: a request entry, a response entry, and a performance entry. The basic format for the execution log is shown below.

```
sequence number~record type~record data
```

The sequence number was automatically generated for each record. The record type was one of the following: "1" for a request log entry, "2" for a response log entry, and "3" for a performance log entry. The record data varied by the record type. The format of the log file is shown in Table 1.

Field Name	Data Type	Description
Sequence Number	Long integer	Auto incrementing value
Record Type	Char	1 = request 2 = response 3 = performance
Record Data when record data = 1 when record data = 2 when record data = 3	String XML String XML String Tilde Delimited String	Varies by Record Type Request as defined by Figure 7 Response as defined by Figure 8 Performance Data String defined below
Performance Data	Delimited String (~)	Contains the following delimited fields
Operation Name	String	Contains the Class Name and Method name of the RPC in class.method format
Image File Name	String	Contains the filename of the image
Image File Size	Integer	Contains the file size in bytes
Selected Surrogate Name	String	Selected Surrogate Name or Local
Execution Latency (C_T)	Integer	Operation latency in milliseconds
Round-Trip Communication (C_C)	Integer	Communication time in milliseconds
Remote Execution Time (C_R)	Integer	Remote execution time in milliseconds
Connected Surrogates	Delimited String (;)	Contains the available surrogates
Connected Surrogate Statuses	Delimited String (;)	Contains the statuses of all connected surrogates

Table 1 – The Execution Log File Format

For record types 1 and 2, the record's data string contained the XML for the request and response, respectively. The performance record field's value contained the following tilde delimited data items. The operation name, the image name, the image size in bytes, the surrogate selected for use, the latency of the execution, round-trip communication time, remote execution time, the connected surrogates, and the status of connected surrogates. This data enabled the performance of the system to be calculated for each of the scheduling algorithms. The overall latency (C_T) was calculated as the summation of the round-trip communication time (C_C) and the operation execution time (C_R). The latency value represents the total time from when a user requested an operation

to be performed until the user received the results of the request. An example of the contents of the execution log file is shown in Appendix B.

Surrogate Discovery

The surrogate discovery subsystem registered and unregistered surrogates with the execution scheduler. When a surrogate connected to jScavenger, the discovery subsystem, shown in Figure 9, registered the surrogate with the execution scheduler and informed the scheduler about the operations the surrogate could perform along with the speed of the surrogate.

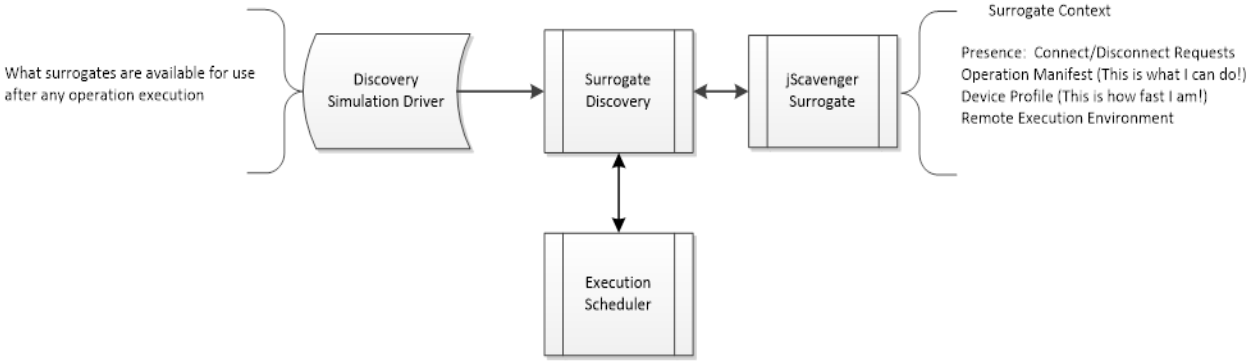

Figure 9 – The jScavenger Surrogate Discovery and Presence Subsystem

Rather than manually orchestrating the availability of devices during the data collection stage of this research, the surrogate discovery module reads a driver file, which managed the availability of surrogates at runtime. The use of the driver file allowed for the same sequence of operations to be executed across all experimental scenarios. Surrogate status was changed after the completion of each operation execution. In this research, surrogates were assumed to be present for the entire duration of a remote execution request and would not be preempted. For example, the following discovery events will change the status of surrogates one, two, and three after the next 3 successive executions of the `CONTRAST` function.

```
RPC CONTRAST SURROGATE-1=ONLINE, SURROGATE-2=OFFLINE, SURROGATE-3=ONLINE;
```

```
RPC CONTRAST SURROGATE-1=OFFLINE, SURROGATE-2=OFFLINE, SURROGATE-3=ONLINE;
RPC CONTRAST SURROGATE-1=ONLINE, SURROGATE-2=COLD-START, SURROGATE-3=OFFLINE;
```

The specification of the Discovery Simulation Driver file format is given in Figure 10 below.

<discovery_file> ::= <discovery_event> { <discovery_event> }

<discovery_event> ::= <tactic_type> <function_name> <surrogate_name> = <state> {, <surrogate_name> = <state> } <terminator>

<tactic_type> ::= RPC

<function_name> ::= <identifier>

<surrogate_name> ::= <node_name>

<identifier> ::= <letter> | <underscore> { <letter> | <digit> | <underscore> }

<node_name> ::= <letter> | <special_characters> { <letter> | <digit> | <special_characters> }

<letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<special_characters> ::= @ | - | <underscore>

<underscore> ::= _

<state> ::= ONLINE | OFFLINE | TRAINING | COLD-START

<terminator> ::= ;

Figure 10 – Discovery Driver File Format Specification

The Application Tactics File

A cyber foraging application was required to provide the jScavenger system with an application tactics file in order to take advantage of cyber foraging services. The tactics file contained the signature for each method that could be offloaded to a surrogate. As shown in Figure 2, the application tactics file was passed to the execution scheduler for use in the offloading decision. The execution scheduler monitored the method executions of a cyber foraging application and if the current method name matched a method in the tactics file, the method was intercepted for remote execution.

A sample application tactic is shown below that describes an RPC called Grayscale that converts an image to grayscale and returns the converted image as a byte array. The format for the tactic file is show in in Figure 11.

```
RCP byte[] ImgeLib.Grayscale (IN byte[] image);
```

The concept of application tactics was presented by Balan, et al. (2003) as a method by which application developers could hand-tune an application rather than have the system enumerate over all the possible ways (most of them infeasible) an application could be partitioned for remote execution. Tactics are the compromise between evaluating every possible combination at run-time, which may be computationally prohibitive, to hard-coding the application so that only one partitioning option was available (Balan, et al., 2003). The authors' also recognized that creating a tactics file was a burden for the application developer; however, since the number of methods that could potentially be remotely executed was typically small compared to the overall size of the application's source code, the added burden was generally manageable. To strengthen the argument for a developer supplied partitioning strategy, Chun, et al. (2010) stated that a hand-tuned partitioning strategy would likely outperform the partitioning generated by an algorithm.

The tactics file approach was selected for jScavenger due to the external nature of the annotations. The external nature of the tactics file was appealing because it allowed the behavior of the cyber foraging application to be changed without source code modifications.

```
<tactic> ::= <tactic_type> <data_type> <function_name> <parameter_list> <terminator>

<tactic_type> ::= RPC

<function_name> ::= <identifier>

<identifier> ::= <letter> | <underscore> { <letter> | <digit> | <underscore> }

<letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S |
             T | U | V | W | X | Y | Z | a | b | c | d | e | f | g | h | i | j | k | l |
             m | n | o | p | q | r | s | t | u | v | w | x | y | z

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<underscore> ::= _

<data_type> ::= void | byte | short | int | long | float | double | boolean | char | byte[] | short[] | int[] |
                long[] | float[] | double[] | boolean[] | char[]

<parameter_list ::= ( <direction> <data_type> <identifier> {, <direction> <data_type> <identifier> } ) <terminator>

<direction> ::= IN | OUT

<terminator> ::= ;
```

Figure 11 - Tactic File Format Specification

The jScavenger Surrogate

A jScavenger Surrogate functioned as a remote operation execution client for a jScavenger Foraging Application Server. Using surrogate devices for remote execution to increase an application's performance was presented by Balan et al. (2002) as a component of the Spectra cyber foraging system and has been a necessary component of virtually all cyber foraging systems to date. The architecture of jScavenger's surrogate was similar to the architecture used by the Scavenger system presented by Kristensen (2010). This section presents the architecture of the jScavenger surrogate and discusses the following components: the presence subsystem, the RPC execution subsystem, the operation manifest, the executable code store, the parameter data repository, and the local execution log. The architectural overview of jScavenger Surrogate is shown in Figure 12.

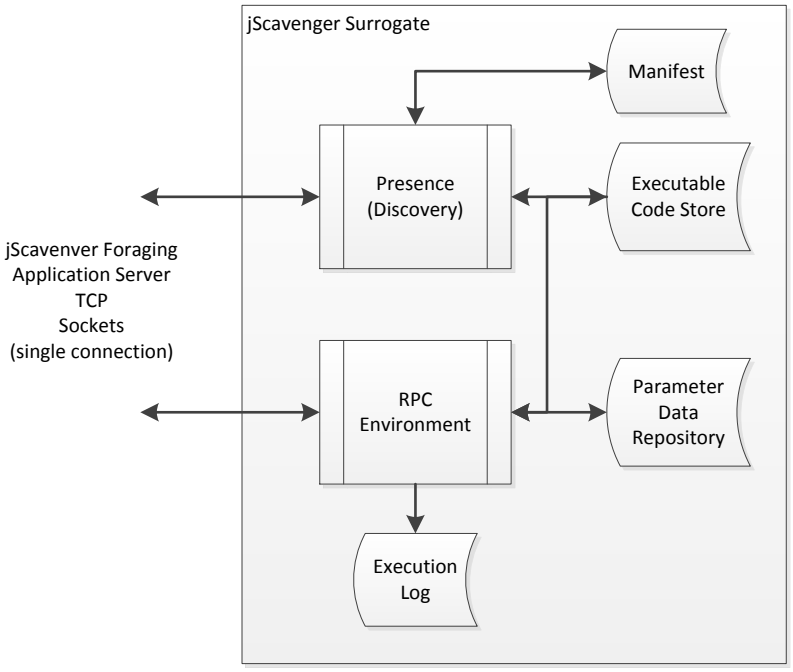

Figure 12 – jScavenger Surrogate Architecture

The Presence Subsystem

The purpose of the presence subsystem was to detect and announce the presence and capabilities of a surrogate to the foraging server. For the purposes of this research, the surrogate presence subsystem was configured to connect to a specific jScavenger foraging server during initialization rather than listening for a broadcast from a local foraging server.

Each surrogate maintained a manifest, which contained details of the operations that it was able to perform. When the surrogate was initialized, each object in the executable code store was examined using Java's Reflection API and the public methods that were present in each class were identified and added to the manifest file. Any public method could be the target of an RPC request. The surrogate updated the manifest when the surrogate was initialized, when new class files were imported, and when class files were removed from the code store. The manifest maintained the type of call, the return

data type, the function name, the parameter direction, the parameter data type, and the parameter name.

An example of a manifest entry is shown below.

```
RPC 567483 byte[] ImageLib.Contrast ( IN byte[] IMAGE );
```

The manifest entry indicates that a remote procedure call having 567,483 JVM instructions that returns a byte array called, “ImageLib.Contrast” is available. The RPC accepts a single byte array as the only parameter. The manifest file specification is shown in Figure 13.

```
<manifest> ::= <manifest_entry> { <manifest_entry> }

<manifest_entry> ::= <call_type> <function_instruction_count> <data_type> <function_name> <parameter_list> <terminator>

<call_type> ::= RPC

<function_instruction_count> ::= <integer>

<data_type> ::= void | byte | short | int | long | float | double | boolean | char | byte[] | short[] | int[] |
                long[] | float[] | double[] | boolean[] | char[]

<function_name> ::= <identifier>

<identifier> ::= <letter> | <underscore> { <letter> | <digit> | <underscore> }

<letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S |
             T | U | V | W | X | Y | Z | a | b | c | d | e | f | g | h | i | j | k | l |
             m | n | o | p | q | r | s | t | u | v | w | x | y | z

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<integer> ::= digit { digit }

<underscore> ::= _

<parameter_list ::= ( <direction> <data_type> <identifier> {, <direction> <data_type> <identifier> } ) <terminator>

<direction> ::= IN | OUT

<terminator> ::= ;
```

Figure 13 – Manifest File Specification

The Remote Execution Environment

The remote execution environment accepted remote procedure call requests from the foraging server, performed the requested operation, and returned a response to the requestor. The format of the request and response documents is shown in Figures 7 and 8, respectively.

When a surrogate received an RPC request, it first read the operationName property from the request document and verified that the operation was available. If the operation was available, then the surrogate extracted the parameter data from the request document and executed the operation. If the operation was not available, an error was generated and a response containing the error information was returned.

The Executable Code Store

The executable code store was the library where RPC operations that a surrogate can offer to foraging applications were stored. For the experiments in this research, the libraries were prepopulated with Java class files for the remote execution environment for use when servicing remote procedure calls.

The Parameter Data Repository

The parameter data repository was a temporary working storage area for the remote execution environment. Any data that was required to be persisted was stored in this area. Data was stored using the requestID followed by the parameter name. For example, if it was necessary to persist the image parameter of the request document shown in Figure 7, a file named 421.IMAGE.dat would be created to hold the parameter value. All files in this storage area were deleted when the surrogate was initialized.

The Surrogate Execution Log File

Every jScavenger surrogate maintained an execution log of each remote execution request received and every response sent. The surrogate's execution log was identical to the foraging server's log with the exception that performance records were not generated. The surrogate logging system only maintained logs for requests and responses. The surrogate log file was called, "surrogate.log."

The jScavenger Cyber Foraging Application

As mentioned earlier, the cyber foraging application utilized in this research was an image manipulation application, which enabled the user to sharpen an image, adjust the contrast of an image, or convert the image to grayscale. The application was an Android application running on a smartphone that allowed the user to select an image and the operation to be performed upon the image. The application was also able to run in autopilot mode by executing a script that automated the image selection and operation requests. A prototype user interface of the testing tool is shown in Figure 14.

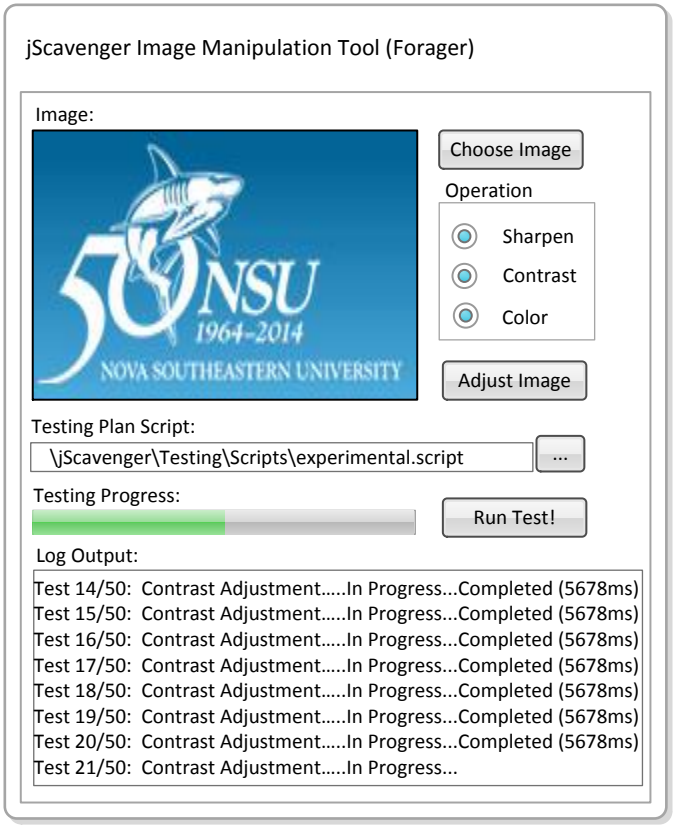

Figure 14 – Image Manipulation Application

If a script file was selected for processing, the script file was expected to contain the image to be processed, the operation(s) to be performed upon the image, and the repetition count. The format of the script file uses name/value pairs and consisted of the

following named attributes. ScriptName, ScriptDescription, ImageName, OperationList, and RepeatCount. The ScriptName attribute contained the free-format name of the script. The ScriptDescription attribute proveded a short description of the script. The ImageName contained the filename of the image to process. The RepeatCount property specified the number of times the operation(s) were to be performed upon the image defined in the ImageName property.

The script file was defined as follows. First, the ScriptName and ScriptDescription must be defined (in-order) on the first two non-commented lines of the script. The parameters ImageList, OperationName, and RepeatCount must follow in-order on the next three consecutive non-commented lines of the script. This parameter set may be repeated to enable multiple operations to be performed in a single test script. A sample script is shown in Figure 15.

```
;Start of script
ScriptName=Experimental.script
ScriptDescription=Experiential Testing Script that sharpens an image, adjusts the contrast,
and adjusts the color image 50 times full-size and 50 times thumbnail (200x200) size.
;
Image=dog.png
OperationList=Sharpen, Contrast, Color
RepeatCount=50
; End of script
```

Figure 15 – Image Manipulation Tool Automation Script Example

When the mobile device takes a picture, the resulting image was stored at /storage/extSdCard/DCIM/Camera/. This location was considered the root folder for the cyber foraging application's images. All input and output from the image manipulation tool defaulted to this location. The image manipulation tool supported the PNG graphic file format.

Operation Profiling

The automatic profiling of operations to obtain the average number of JVM instructions that were expected to be executed when the operation was invoked was achieved using a CFG. Java bytecode was transformed into a CFG graph, which was traversed to quantify the number of JVM instructions that would be executed when the method was invoked. The use of bytecode was preferred over source code because access to the source code could not be guaranteed.

Albert et al. (2007) utilized CFGs in one step of an algorithm used to determine the cost relationship between methods and their input parameters. The authors' work utilized CFGs to transform unstructured bytecode, which was difficult to analyze directly in part due to the use of the *goto* statement, into a traversable graph data structure that was suitable for static code analysis. The use of CFGs in this research was orthogonal to the work of Albert et al. (2007) in the sense that this research was concerned with obtaining a generic cost estimate of the operation's execution for use as a heuristic. Although interesting, this research does not require the general relationship between the input parameters and the amount of work performed by the operation. The high-level algorithm used to profile operations in jScavenger follows:

1. Convert the class containing the operation into a CFG.
2. Generate a list of all possible paths from start to exit for the CFG.
3. Calculate the average number of operations across all execution paths in the list.

Consider the simple Java program shown in Figure 16. The entire program consists of 12 statements, but because of the *if* statement, not all of the statements will be

executed when the program runs. Figure 17 shows the same program converted into a CFG. There are two execution paths from start to exit in this program. Each block contains statements, which are guaranteed to be executed. In Figure 17, the True and False arrows from the decision statement define a fork in the execution path. To determine the number of statements that could be executed, the number of statements contained in each block for each execution path will be summed. An average will then be calculated across all paths to provide a high-level approximation of the number of statements that will be executed. Using this technique, the estimated number of statements that will be executed when this program runs is 9 statements. This number represents the estimated cost of the operation.

```
int a = 5;
int b = 10;
System.out.println("A = " + a);
System.out.println("B = " + b);

if( a > 5 )
{
  a++;
  b--;
}
else
{
  a--;
  b++;
}
System.out.println("A = " + a);
System.out.println("B = " + b);
```

Figure 16 - Simple Java program

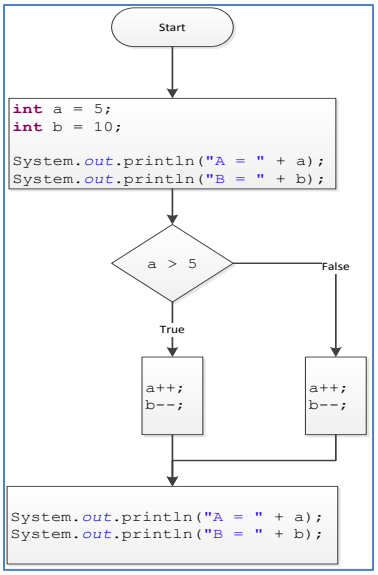

Figure 17 – Control Flow Graph of the Program in Figure 8

The example above utilized Java source code to illustrate the CFG based approach to calculate the number of statements that could be executed. The actual approach utilized Java bytecode. The JVM instructions that would be executed by the JVM when the Simple Java Program was executed are shown in appendix A. Applying the technique discussed above to the output of the *javap* command yielded approximately forty-six JVM instructions that would be executed when the program runs.

The Apache Commons Byte Code Engineering Library (BCEL), which provided methods that enabled a Java program to inspect and manipulate Java bytecode, was utilized to extract the Java instructions from bytecode to calculate the average number of instructions at run-time. In future work, it would be advantageous for the Java compiler to generate this value during the compilation process and store the value as a method annotation.

Device Profiling

To support the experimental scheduling algorithm, each device in the jScavenger system was profiled to obtain a measure of how powerful it was for executing operations.

In jScavenger, the speed of the surrogate was the number of JVM instructions the device demonstrated it could execute in one second. Binder and Hulaas (2006) first proposed using Java bytecode instruction counting as a method to obtain a cross-platform method of expressing CPU utilization rather than using the more traditional method of using CPU seconds. The benefit of using bytecode counting was that it effectively removed the variability introduced by the underlying hardware that influenced the CPU metric (Binder & Hulaas, 2006). This approach enabled devices to be rated by a common metric that does not have to be adjusted based upon the platform upon where the code was executed.

Kafaie, et al. (2011) utilized a similar approach by rating an operation's speed on a specific device by how many input data elements could be processed by an operation in one second. This approach provided a metric that defined how an operation performed on a specific device; however, the approach requires that each operation be profiled per device prior to use. Kristensen and Bouvin (2010) utilized the third-party benchmarking suite NBench in the Scavenger system to provide a common rating of a device's strength. The Scavenger system utilized the benchmark score as a device's strength indicator, which allowed devices to be compared with one another. Both of these approaches provided a strength or speed indicator of the device; however, both approaches required manual developer support and an offline profiling session before the operations/devices were available for use.

The approach to profiling devices in this research aimed to eliminate the offline profiling phase by replacing it with an online profiling phase that was integrated into the application. This was achieved by profiling both the jScavenger foraging server and the jScavenger surrogate using the operation profiling method discussed previously to obtain

the number of instructions expected to be executed during the initialization of the jScavenger Foraging Application Server and the jScavenger Surrogate, respectively. During initialization, each application measured the time required to perform the initialization process and determined the speed of the current device using the number of instructions obtained by profiling the initialization code. The resulting JVM instruction execution speed was used to express the overall speed of the device.

The Testing Environment

The testing environment consisted of a wireless mobile device, a wireless access point, a switch, a DHCP server, and 5 surrogate devices. The testing environment is shown in Figure 18. The mobile device was a Samsung Galaxy SIII running Android 4.4, the wireless access point was a Linksys WAP54G, the switch was a Cisco 2950-12 12-port switch, and the 5 computers acting as surrogates were configured as follows:

1. HP D530, Intel Pentium 4, 2.8 GHz, 1 GB RAM, Ubuntu 12.04 x86
2. Power Mac G5 – Motorola PowerPC 970 G5, 1.6 GHz, 512 MB RAM, OS X 10.5.8
3. Compaq 5700T, Intel Pentium 3, 550 MHz, 512 MB RAM, Windows XP
4. Mac Pro Dual Intel Xeon Quad Core Processors, 2.8 GHz, 4 GB RAM, OS X 10.8.5
5. HP 6000 Pro, Intel Core2 Quad Core, 2.83 GHz, 8GB RAM, Windows 7 x64

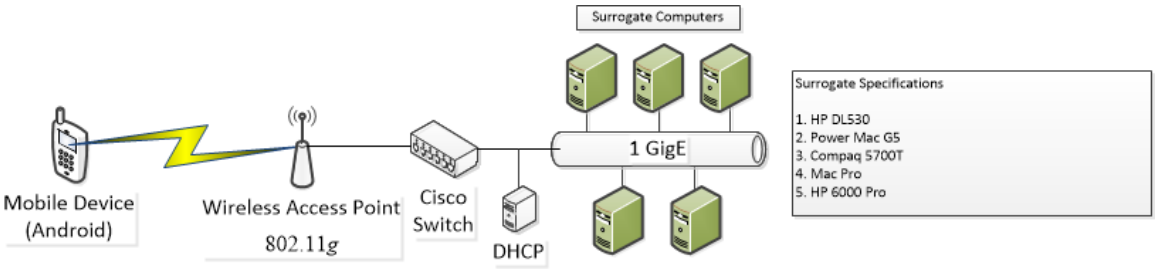

Figure 18 – jScavenger Test System Architecture

The test network was configured as follows. The wireless access point (WAP) was configured with a SSID of 'cyberforaging' and secured with WPA2 Personal encryption. The test network was available in an open environment where the wireless signal could not be masked. To prevent outside devices from interfering with the testing, Media Access Control (MAC) address filtering was enabled on the WAP so that only wireless devices included in the test were able to connect to the network. The network was not connected to either the internet or the enterprise network so a DHCP server was installed on the network to assign IP addresses based on each device's unique MAC address. Only devices that were configured in the DHCP server's configuration file were provided with an IP address.

Each surrogate computer was configured to run Java version 7.0 SE. Each surrogate client installed was configured to connect to the foraging server running on the mobile device upon initialization. Each surrogate was configured with a secure shell server so the surrogates could be remotely administered.

Performance Evaluation

Three experiments were conducted to measure the performance of the new scheduling algorithm proposed in this research. The first experiment measured the performance of the cyber foraging application with the historical-based prediction algorithm. The second experiment measured the performance of the cyber foraging

application with the experimental algorithm. The third experiment measured the performance of the cyber foraging application with the random scheduling algorithm.

Each experiment consisted of 4 scenarios, which collected data for specific operating conditions. The scenarios were disconnected operation, a saturated environment, a slowly churning and building environment, and a quickly churning environment. The structure of the each experiment is shown below followed by the description of each scenario.

1. Experiment #1 – Historical Algorithm
    a. Scenario #1 – Disconnected Operation
    b. Scenario #2 – Saturated Environment
    c. Scenario #3 – Slowly Churning Environment
    d. Scenario #4 – Quickly Churning Environment
2. Experiment #2 – Experimental Algorithm
    a. Scenario #1 – Disconnected Operation
    b. Scenario #2 – Saturated Environment
    c. Scenario #3 – Slowly Churning Environment
    d. Scenario #4 – Quickly Churning Environment
3. Experiment #3 – Random Algorithm
    a. Scenario #1 – Disconnected Operation
    b. Scenario #2 – Saturated Environment
    c. Scenario #3 – Slowly Churning Environment
    d. Scenario #4 – Quickly Churning Environment

The disconnected operation scenario tested the system in an environment where there are no surrogates available. In this situation, all operations were executed locally, thus providing a baseline measurement of the cyber foraging application performance without cyber foraging assistance.

The saturated environment scenario tested the system in an over-provisioned environment where multiple surrogates were available. The purpose of this scenario was to measure the performance in a static system where there was no surrogate churn. This scenario was similar to a device being used at an individual's home or workplace, where the presence of other devices can be predicted in advance and rarely changes.

The slowly churning environment scenario exercised the system in an environment where there were initially no connected surrogates and surrogates were slowly added to the system until all five surrogates were available for use. The purpose of this scenario was to measure the performance of the scheduling algorithm in an environment where changes were slow but constant. This scenario started out with no surrogates available and a new surrogate was added every 5 operations. This scenario parallels a business or social meeting place where people arrive sporadically and once present do not leave for an extended period.

The quickly churning environment scenario exercised the system in an environment where initially no surrogates were available. Surrogates were then simultaneously added and removed from the system after every few operations. The purpose of this scenario was to measure the performance of the scheduler in an environment where changes were fast-paced and constant, with possibly multiple surrogates arriving and departing simultaneously. This specific scenario started with no

surrogates online. After a random number of operations were performed (between 1 and 5), one surrogate was randomly selected to join the system and one currently connected surrogate was disconnected (if applicable). In this scenario, all surrogates were considered new to the system and training state will not be retained between a surrogate disconnection and a surrogate reconnection. This scenario parallels use in a public place such as a café or an airport terminal, where devices and their owners are highly mobile.

This research replicated the mechanics of the benchmarking approach used in Scavenger, where a series of image operations were sequentially performed on an image 50 times, once using a thumbnail representation of the image (200x200) and once using the full-sized image (Kristensen & Bouvin, 2010). According to Kristensen & Bouvin (2010), it is common for a series of operations to be performed upon an image before it was published. In the experiments outlined above, the image operations to be performed upon an image were to sharpen the image, adjust the contrast of the image, and to convert the image to grayscale. The images used were high-definition full-color pictures taken with an 8 Megapixel camera at a resolution of 3264x2448. The thumbnail images were derived from the full-size images at run-time.

The Data Collection Process

The execution logs created while running the experiments was named according to the currently running experiment and scenario. All log files were named based upon the experiment and scenario being conducted as shown in Table 2. The log files collected during this process was formatted according to the execution log file format shown in Table 1.

Experiment	Scenario	Log File Name
Historical	Disconnected Operation	Historical.disconnected.log
Historical	Saturated Environment	Historical.saturated.log
Historical	Slowly Churning Environment	Historical.slowchurn.log
Historical	Quickly Churning Environment	Historical.quickchrun.log
Experimental	Disconnected Operation	Experimental.disconnected.log
Experimental	Saturated Environment	Experimental.saturated.log
Experimental	Slowly Churning Environment	Experimental.slowchurn.log
Experimental	Quickly Churning Environment	Experimental.quickchrun.log
Random	Disconnected Operation	Random.disconnected.log
Random	Saturated Environment	Random.saturated.log
Random	Slowly Churning Environment	Random.slowchurn.log
Random	Quickly Churning Environment	Random.quickchrun.log

Table 2 – Log File Naming by Experiment and Scenario.

The Data Analysis Process

The data captured by performing the experiments was imported into a Microsoft Access database, labeled by experiment, and processed to complete the following analysis. The analysis consisted of a high-level overview of the results, an analysis of the historical algorithm, an analysis of the experimental algorithm, a brief analysis of the random algorithm, and a discussion of the combined analysis of all three algorithms.

First, a high-level overview of the experiments was presented. This overview included the execution time of each scheduling algorithm. The results were graphed to present a high-level performance overview of each scheduling algorithm. Next, the performance of each scheduling algorithm was graphed to compare the performance of each operational scenario. Finally, the detailed execution time of each experimental scenario was presented, which included both the local and remote execution time.

The historical scheduling algorithm was examined by reviewing the surrogate training and selection activity for each of the scenarios. For each scenario, a graph was generated that presented the both the local execution time and the remote execution time for each scenario. This enabled the surrogate selection and training activities to be compared against the expected surrogate selection and training activities.

The experimental scheduling algorithm was analyzed by first presenting the surrogate configuration profiles along with the speed ranking of each surrogate. A table was generated from the random saturated scenario that included multiple operation executions on each surrogate to obtain actual execution metrics. The data were sorted by execution time to compare the calculated speed vs. the actual execution time required by each operation. Each scenario was then graphed in the same fashion as the historical scheduling algorithm so that the surrogate selection could be compared with the expected results.

The random scheduling algorithm was not graphed due the random nature of surrogate selection. To assess this algorithm, the overall execution time for each scenario was compared against the historical and experimental algorithms.

All three scheduling algorithms were compared against each other by superimposing all of the performance graphs for each scenario. The resulting graphs illustrated the performance of each algorithm by operational scenario. This allowed the performance to be reviewed by operation.

Data Verification

To ensure that the HotSpot JVM was not influencing the results, each JVM was configured with the –Xint run-time parameter to disable just in time (JIT) compilation.

On the Android platform, this was accomplished by using the android:vmSafeMode="true" property of the Android application manifest configuration file. Additionally, the JVM were configured to display compilation messages using the –XX:-PrintCompliation parameter to ensure that the JVM did not perform optimizations that would influence the execution time of the operations.

Resources

The computing resources required to complete this dissertation consisted of a development machine, Java application development software, Android application development software, a wireless access point, a switch, networking accessories, an Android-based cell phone, and 6 network ready personal computers capable of running Java 7.0.

The development environment consisted of Java 7.0 (http://java.oracle.com), the Android Developer Tools (ADT) bundle (http://dl.google.com/android/adt/adt-bundle-windows-x86_64-20131030.zip), and the Eclipse Java IDE (http://www.eclipse.org/downloads/). The test network infrastructure consisted of a Linksys WAP54G, and a Cisco 2950-12 switch with the required configurations and cabling necessary to create an isolated local area network.

Summary

This research developed and implemented an enhanced historical-based prediction algorithm. This algorithm utilized estimation for surrogate selection during the cold-start state of a cyber foraging application until the historical-based prediction algorithm accumulated enough execution history to make predictions. To provide an environment

where the new algorithm could be evaluated, a Java-based cyber foraging system, called jScavenger, was developed.

Three scheduling algorithms were utilized by jScavenger to test the performance of each algorithm. In addition to the experimental algorithm, the additional scheduling algorithms were a historical-based algorithm and a random-based algorithm. The experimental algorithm utilized a heuristic based upon the rate at which a surrogate demonstrated that it could execute JVM instructions and the number of JVM instructions contained within an operation to choose a surrogate when a historical prediction was unavailable. The historical-based algorithm utilized historical predictions to select surrogates and when a surrogate was unable to provide a prediction, a training session was initiated to obtain historical measurements. The random-based algorithm chooses surrogates based upon a random number generator.

Each scheduling algorithm was utilized in 4 testing scenarios, each of which collected data for specific operating condition. The scenarios were disconnected operation, a saturated environment, a slowly churning and building environment, and a quickly churning environment. The disconnected operation scenario tested the system in an environment where there are no surrogates available. The saturated environment scenario tested the system in an over-provisioned environment where multiple surrogates were available. The slowly churning environment scenario exercised the system in an environment where there were initially no connected surrogates and surrogates were slowly added to the system until all five surrogates were available for use. The quickly churning environment scenario exercised the system in an environment where initially no

surrogates were available. Surrogates were then simultaneously added and removed from the system after every few operations.

The results of each scheduler's performance of each operational scenario were analyzed independently, then against each of the other two scheduling algorithms. This allowed for the scheduler's performance with each operational scenario to be verified and then the performance to be compared with the other algorithms.

Chapter 4

Results

Introduction

Three experiments were conducted based upon the methodology described in Chapter 3. The results of the experiments are discussed in the following sections.

- Overview of the Experimental Results
- Experiment 1 – Historical Scheduling Algorithm
- Experiment 2 – Experimental Scheduling Algorithm
- Experiment 3 – Random Scheduling Algorithm
- A Performance Comparison of the Experiments

Overview of the Experimental Results

The execution time required to complete all three experiments is shown in Figure 19. The overall execution time is presented for each experiment grouped by the time spent executing operations locally (C_T) and the time spent executing operations remotely (C_R). Additionally, the time required for the experiments to be run in disconnected from the network is also presented as a baseline measurement. The network communication overhead is not included in the calculations.

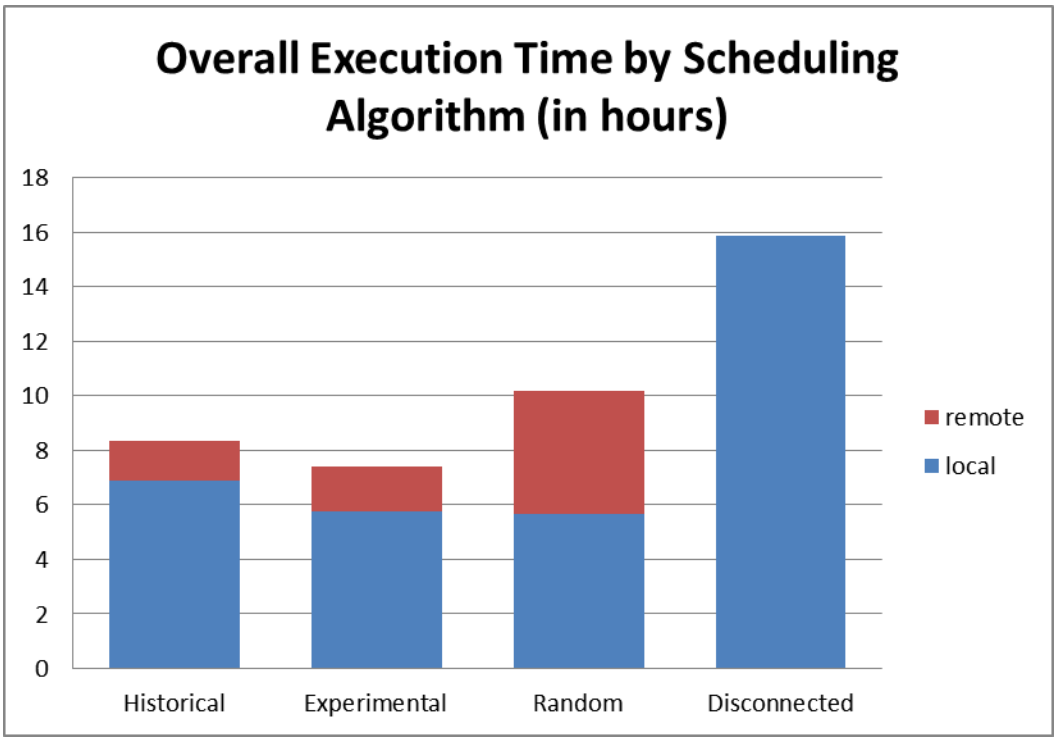

Figure 19 – Overall Execution Time by Scheduling Algorithm

Overall, the experimental scheduling algorithm required 7.38 hours to execute the 1200 operations in the testing scenario, the historical scheduling algorithm required 8.25 hours to execute the same set of 1200 operations, and the random scheduling algorithm required 10.2 hours to complete the operations. In disconnected mode, the same 1200 operations took 15.84 hours to complete running solely on the local device. Figure 20 presents the performance of each scenario by experiment.

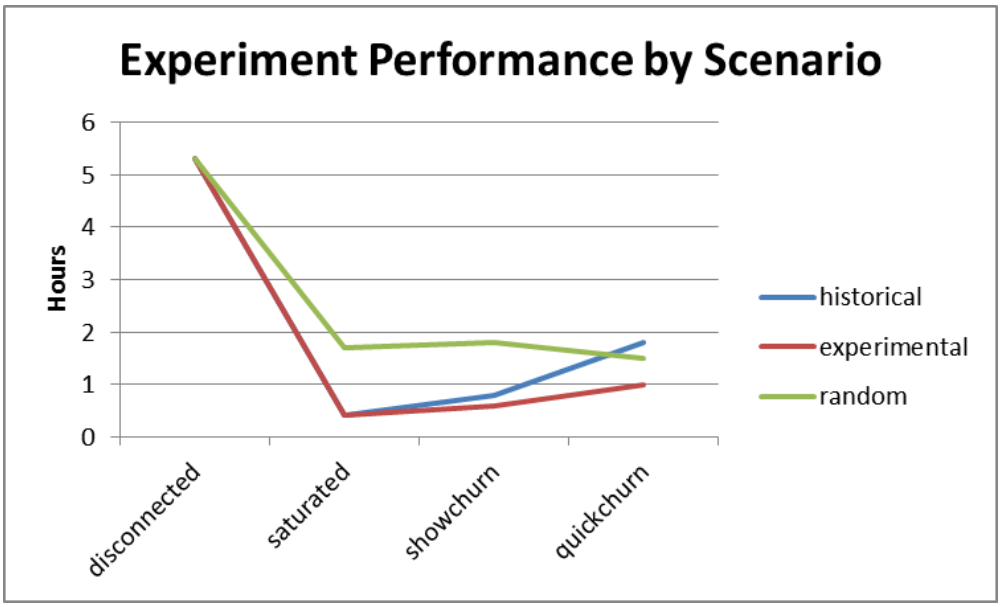

Figure 20 – Experiment Performance by Scenario

Table 3 presents the summarized data for each experiment broken down by scenario. In addition to providing data by scenario, the breakdown of the local and remote execution times are shown along with the total time.

Overall Performance By Experiment and Scenario				
Experiment	Scenario	Local Operation Execution Time (C_T hours)	Remote Operation Execution Time (C_R hours)	Total Time $C_T + C_R$ (hours)
Historical	Disconnected	5.28	0.00	5.28
	Saturated	0.00	0.43	0.43
	Slow Churn	0.32	0.43	0.75
	Quick Churn	1.18	0.61	1.79
Total Time		**6.78**	**1.47**	**8.25**
Experimental	Disconnected	5.28	0	5.28
	Saturated	0.00	0.44	0.44
	Slow Churn	0.21	0.43	0.64
	Quick Churn	0.28	0.74	1.02
Total Time		**5.77**	**1.61**	**7.38**
Random	Disconnected	5.28	0	5.28
	Saturated	0	1.65	1.65
	Slow Churn	0.31	1.48	1.79
	Quick Churn	0.08	1.40	1.48
Total Time		**5.67**	**4.53**	**10.20**

Table 3 – Overall Performance by Experiment and Scenario

Figure 21 shows the execution results of performing all three image operations on a disconnected surrogate using the full-sized image. Since no cyber foraging was involved, this result is common to each of the three experiments. The graph clearly shows the execution for each operation and transitions between the operations running solely on the local device.

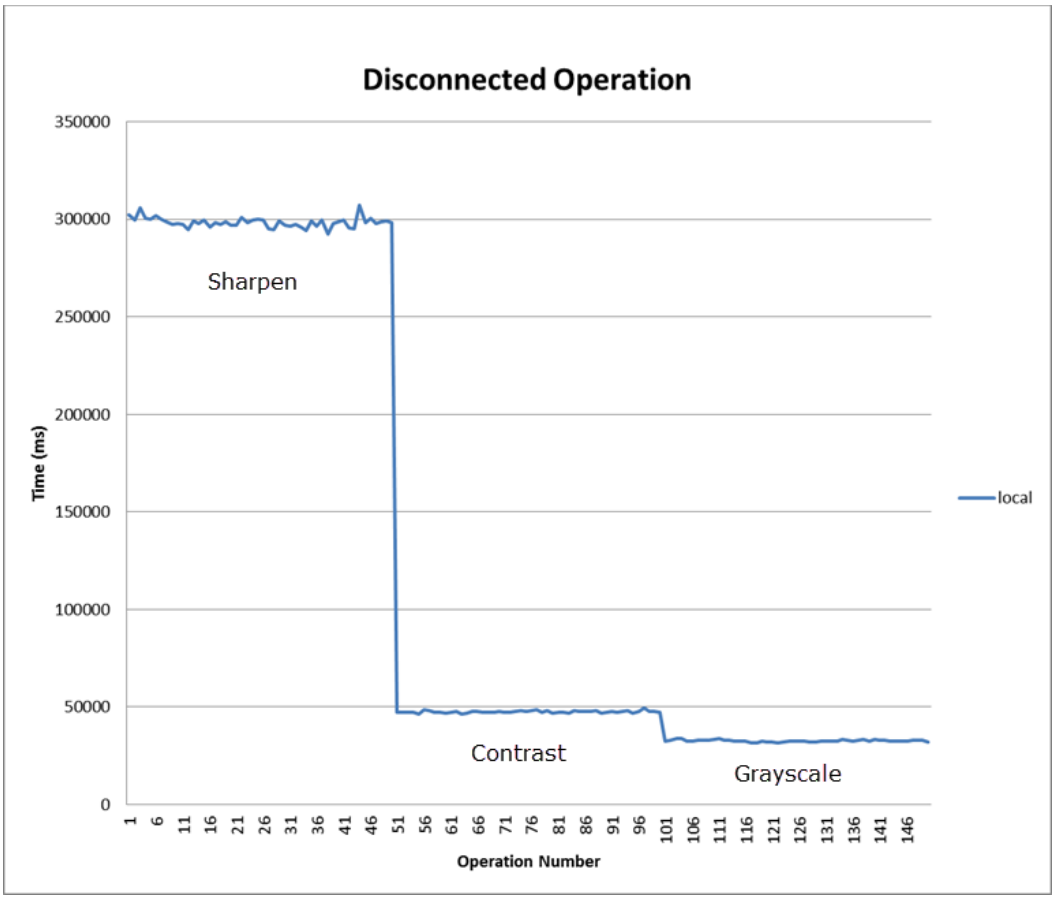

Figure 21 – Disconnected Operation Performance

Experiment 1 – Historical Scheduling Algorithm

Figure 22 shows the results of the historical scheduling algorithm in the saturated scenario where the image manipulation operations were executed on the full-size image. The graph clearly shows the expected local executions at the transitions between operations while the surrogates are trained. Training occurs at the first operation because

no operations have been performed on any of the surrogate devices. Training occurs again at operation 51 because the operation changes from sharpening to contrast and execution history for the contrast operation do not exist. The final training session occurs at operation count 101 when the operation changes from contrast to grayscale due to a lack of execution history. As expected, the scheduler quickly determines the beneficial surrogate to utilize based upon the training. Overall, there are 15 individual training sessions during this experimental run.

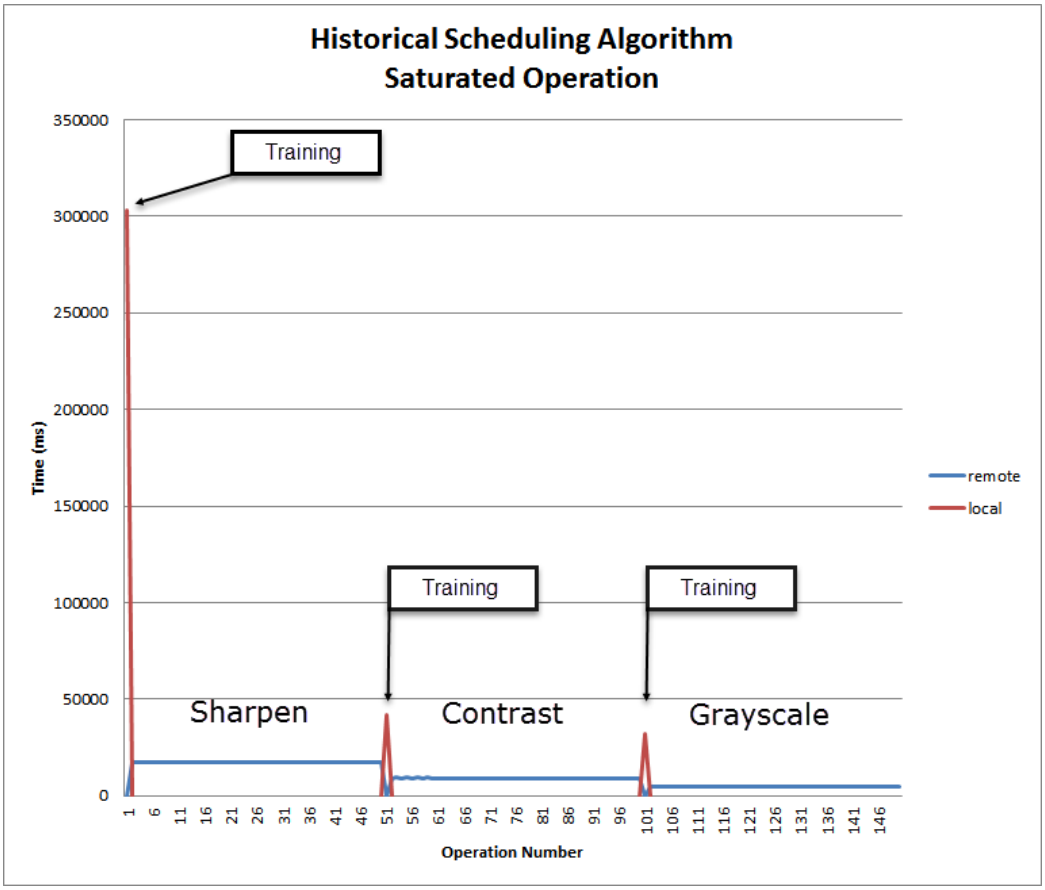

Figure 22

Figure 23 shows the results of the historical scheduling algorithm in the slowly churning scenario, where image manipulation operations were executed on the full-size image. The graph shows the expected local executions at the start of the experiment

while no surrogates are available and again while the initial training is performed once a surrogate becomes available. At operation 13, a new, more powerful surrogate becomes available. The system spawns a background thread to negotiate and train the new surrogate (not shown). At operation 14, the scheduler selects the newly available surrogate and begins offloading operations to that surrogate. New surrogates continue to arrive during the remainder of the test, but they are all less powerful than the currently selected surrogate. The scenario repeats itself for the subsequent operations and the same surrogate switch occurs again at operations 64 and 115 (denoted by the red asterisk). For this scenario, there are 14 individual training sessions during this experimental run.

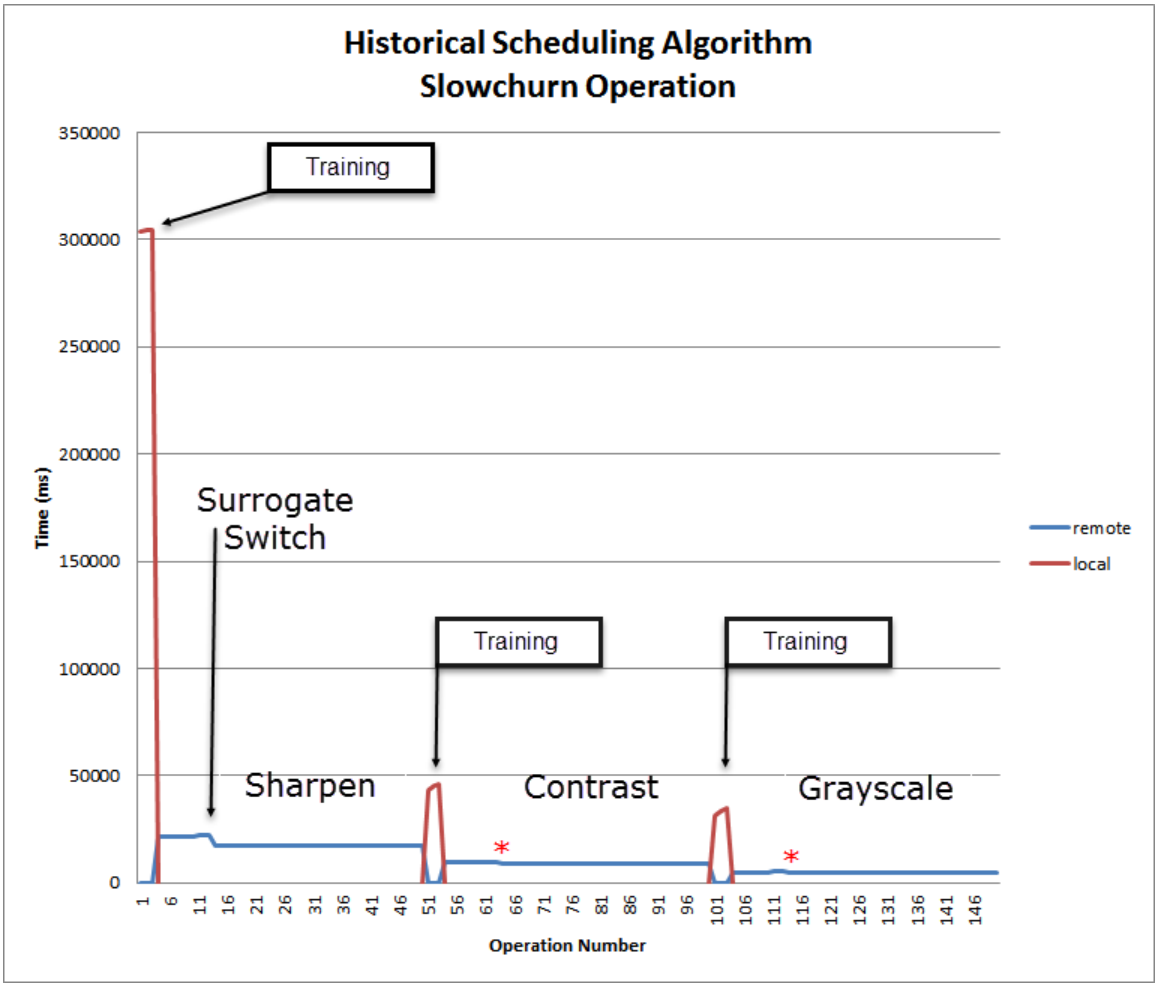

Figure 23 – Historical Scheduling Algorithm Slowly Churning Operation Performance

Figure 24 shows the results of the historical scheduling algorithm in the quickly churning scenario where the image manipulation operations were executed on the full-size image. The graph shows the expected local executions at the start of the experiment when there are no surrogates available and while the initial training is performed. Since the availability of the surrogates is random and they connect and disconnect frequently, the amount of local execution is noticeably higher due to the churn. At operation 19, an abnormally long running remote execution is shown. This spike, although large, is still faster than the observed local executions. It is also noteworthy because it represents a missed opportunity. At operation 18, a new and more powerful surrogate arrived, but it was not selected for remote execution because training had not yet been completed. Overall, the training overhead for this scenario involves 64 individual surrogate training sessions and 68% of all operation executions occur when there is a minimum of one surrogate in training.

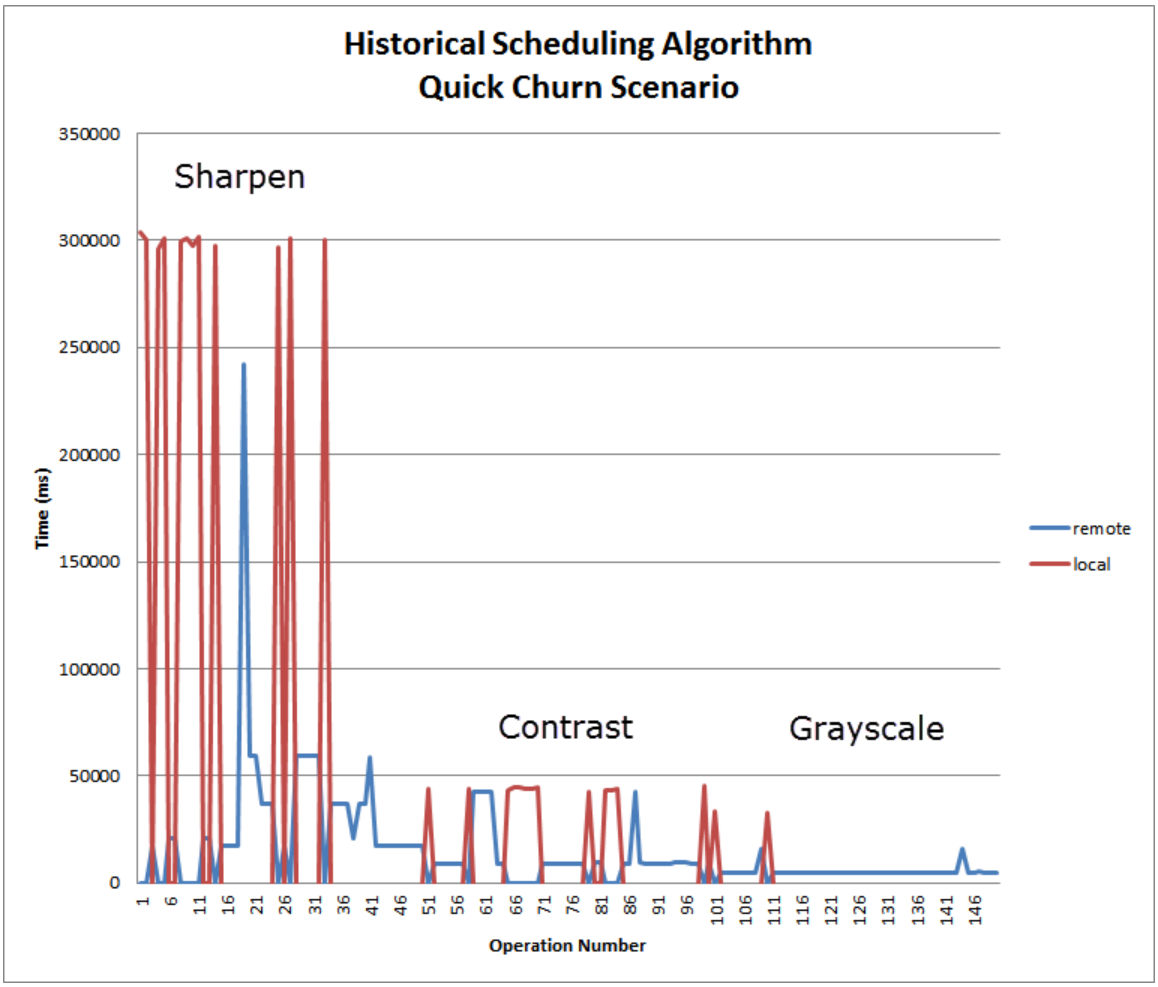

Figure 24 - Historical Scheduling Algorithm Quickly Churning Operation Performance

Experiment 2 – Experimental Scheduling Algorithm

The experimental scheduling algorithm chooses a surrogate based upon the average speed at which a surrogate can process Java instructions. The average number of Java bytecode instructions obtained by traversing the CFG graph for the operations contrast, grayscale, and sharpen is 227, 174, and 285, respectively. The initialization routine for the surrogate client consists of an average of 930 JVM instructions to be executed upon startup. A profile of each surrogate and the surrogate's calculated speed rating is shown in Table 4.

Surrogate Profile				
Surrogate	Speed Rating	CPU	CPU Speed	RAM
S-1	22	1 Intel Pentium 4	2.8GHz	1 GB
S-2	9	1 PowerPC 970	1.6GHz	512 MB
S-3	4	1 Intel Pentium 3	550 MHz	512 MB
S-4	44	2 Intel Quad Core Xenon	2.8GHz	4 GB
S-5	58	1 Intel Core2 Quad Core	2.83GHz	8 GB
local	16	1 ARM Cortex-A9 Quad Core	1.4GHz	1 GB

Table 4 – Surrogate Profile

Every surrogate was utilized in the random scheduling algorithm execution, which enabled the average actual execution time to be measured for each operation on every surrogate device. The average operation execution time for each surrogate along with the surrogate's speed ranking is shown in Table 5. The results are discussed next.

As expected, the contrast operation's actual execution time on each surrogate followed the speed ranking for the decision making with S-5 being the fastest surrogate for the contrast operation to surrogate S-3 being the slowest surrogate for the contrast operation. The speed rankings for the surrogate using the speed heuristic were consistent with the observed execution times for the operation. The grayscale operation ranked surrogate S-5 as the fastest surrogate to S-3 being the slowest surrogate. The speed

rankings for the surrogate using the speed heuristic were consistent with the observed execution times for the operation with one exception. For the grayscale operation, the local execution speed was ranked higher than the observed execution placed it in the rankings. The sharpen operation ranked surrogate S-5 as the fastest surrogate to local execution being the slowest surrogate. The speed rankings for the surrogate using the speed heuristic were consistent with the observed execution times for the operation with the exception of 2 surrogates. For the sharpen operation, both surrogates S-2 and S-3 were ranked slower than their observed executions. These inconsistencies offer opportunities for future research that might include hardware variation between machines.

Average Surrogate Operation Execution Time				
Full-Size Image				
Operation	Surrogate	Surrogate Speed	Operation Speed	Execution Time (ms)
Contrast	S-5	**58**	**3.90**	**9,247**
	S-4	**44**	**5.16**	**9,521**
	S-1	**22**	**10.32**	**43,199**
	local	**16**	**14.19**	**47,570**
	S-2	**9**	**25.20**	**64,476**
	S-3	**4**	**56.75**	**122,996**
Grayscale	S-5	**58**	**3.00**	**4,770**
	S-4	**44**	**3.96**	**5,068**
	S-1	**22**	**5.77**	**15,887**
	S-2	**9**	**14.11**	**22,568**
	local	**16**	**7.94**	**32,646**
	S-3	**4**	**31.75**	**51,926**
Sharpen	S-5	**58**	**4.91**	**17,729**
	S-4	**44**	**6.48**	**20,977**
	S-2	**9**	**31.67**	**36,658**
	S-1	**22**	**12.96**	**59,308**
	S-3	**4**	**71.25**	**210,126**
	local	**16**	**17.81**	**298,367**

Table 5 – Surrogate Performance

The results of the experimental scheduling algorithm in the saturated scenario are presented in Figure 25. In this scenario, the image manipulation operations were performed upon the full-size image where all the surrogates are online and available. As expected, the scheduler quickly determined the beneficial surrogate to utilize without performing local executions.

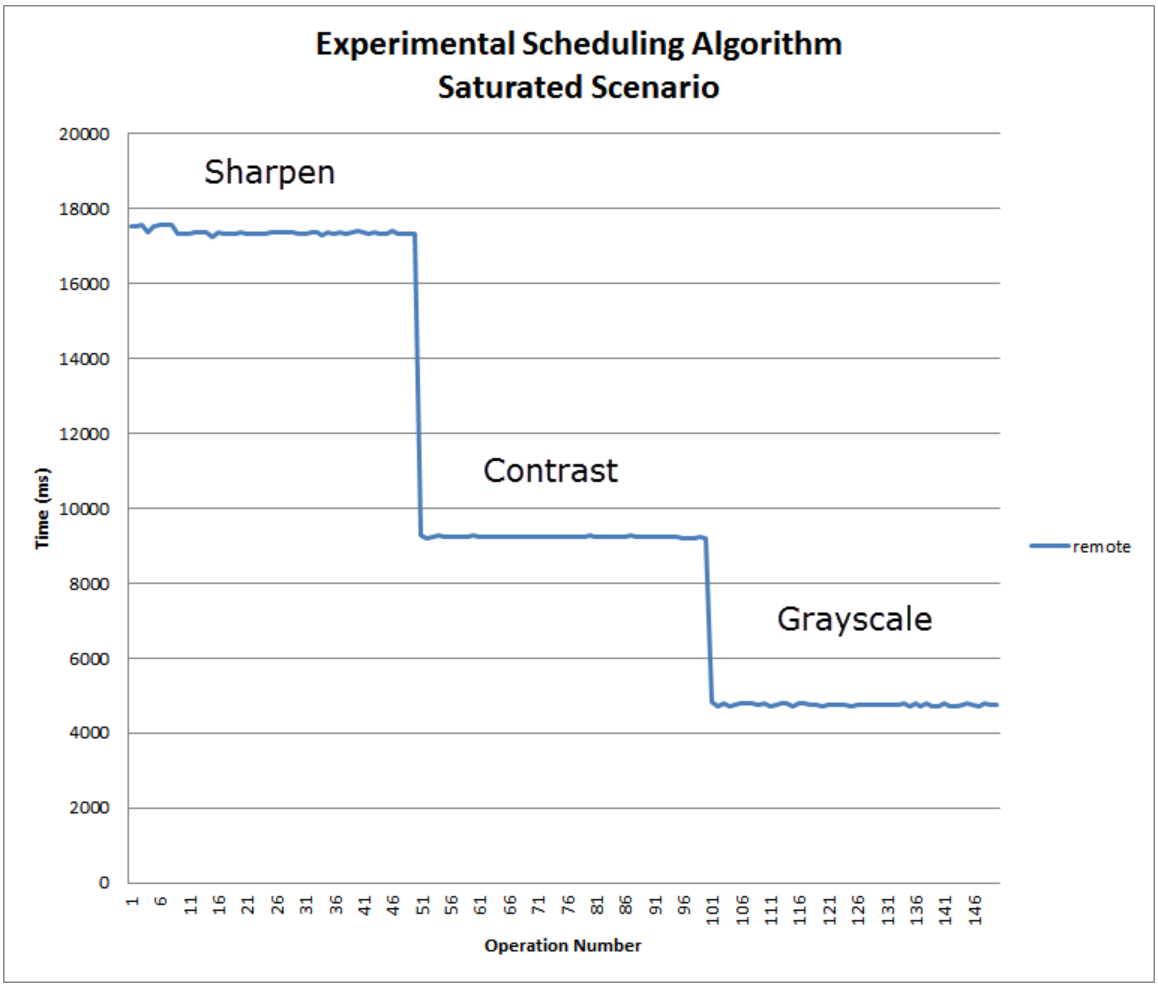

Figure 25 - Experimental Scheduling Algorithm Saturated Operation Performance

Figure 26 shows the results of the experimental scheduling algorithm in the slowly churning scenario, where the image manipulation operations were executed on the full-size image as surrogates slowly join the system. The graph shows the expected local executions at the start of the experiment when no surrogates are online. At operation 13, a more powerful surrogate became available and was immediately utilized by the scheduler to execute operations. New surrogates continued to arrive during the remainder of the test, but they were all less powerful than the current surrogate was. The scenario repeats itself for the subsequent image operations and the same surrogate switch occurs again at operations 63 and 113 (denoted by the red asterisk).

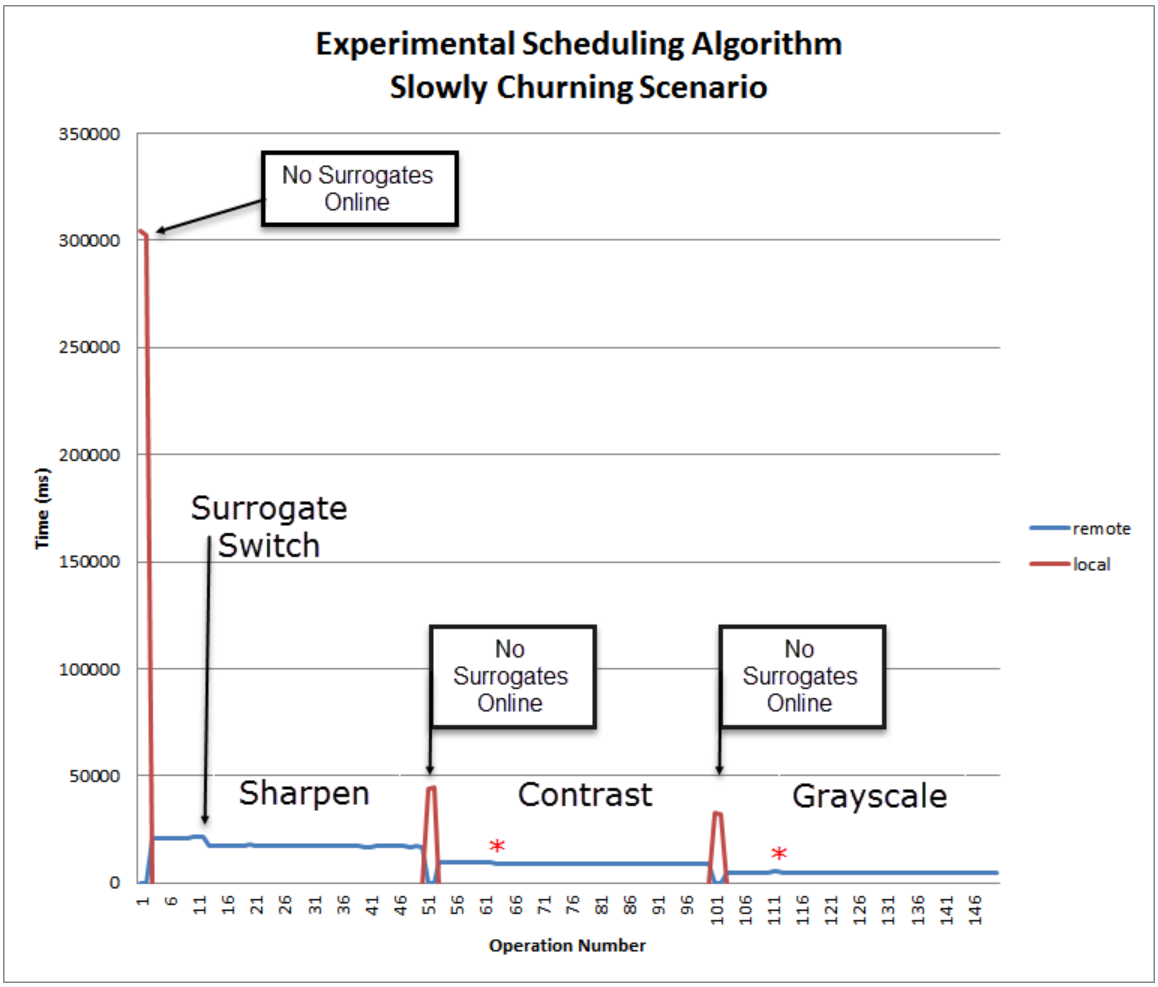

Figure 26 - Experimental Scheduling Algorithm Slowly Churning Operation Performance

Figure 27 shows the results of the experimental scheduling algorithm in the quickly churning scenario, where the image manipulation operations were executed on the full-size image. The graph shows the expected local executions at the start of the experiment when no surrogates are online. Overall, the experimental algorithm produced the best performance over the historical and random algorithms. While the experimental scheduling algorithm picked the proper surrogate based upon the rankings, the performance of three surrogates that were chosen for the contrast and grayscale operations show that in hindsight the choices could have been better.

For the sharpen operation, at operation 4 the local surrogate was chosen by the experimental scheduling algorithm for the contrast operation over surrogate S-3 because it is ranked higher. In reality, surrogate S-3 demonstrated better performance in the post-execution review. For operations 21through 24, surrogate s-1 was chosen over surrogate s-2 where in reality, surrogate s-2 demonstrated better performance. At operation 33, the local device was chosen over surrogate s-2 when s-2 demonstrated better performance. For operations 34-36 and 39-40, s1 was chosen when s2 would have been preferred. For the grayscale operation, the scheduler chose surrogate s-1 over s-2 at operation 109 when s-2 would have provided better performance.

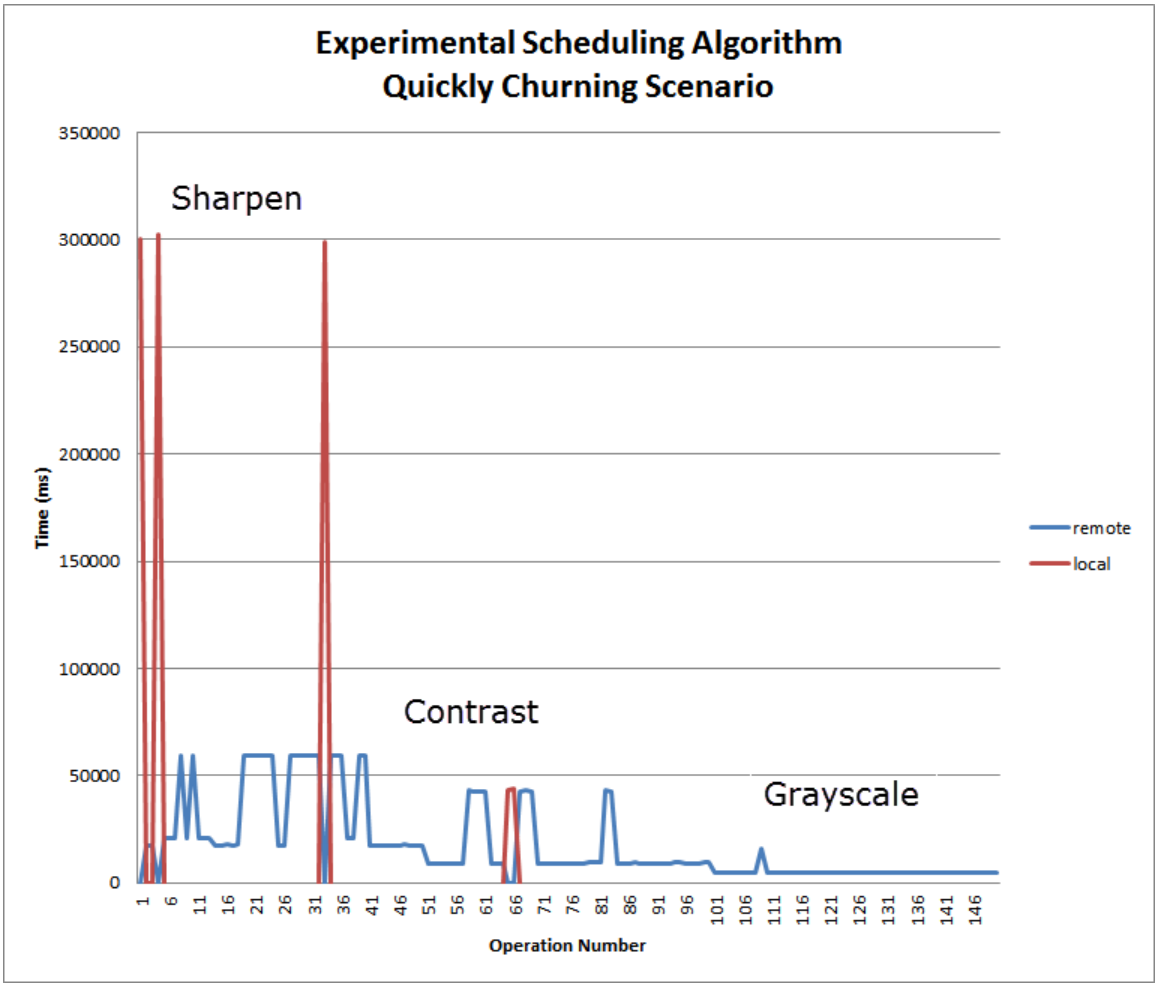

Figure 27 - Experimental Scheduling Algorithm Quickly Churning Operation Performance

Experiment 3 – Random Scheduling Algorithm

The random scheduling algorithm randomly selected a surrogate to perform an operation from the pool of available surrogates. This algorithm assumes that offloading is always beneficial.

The disconnected scenario performed all operations locally due to the lack of available surrogates taking 5.28 (Table 3) hours to complete all 1200 operations. The saturated algorithm performed all 1200 operations on surrogate devices taking 1.65 hours to complete compared to the 0.53 hours for the historical algorithm and 0.44 hours for the experimental algorithm. The slowly churning scenario executed all but 9 operations on surrogate devices and took 1.79 hours to complete compared to the 0.75 hours for the historical algorithm and 0.64 hours for the experimental algorithm. The quickly churning scenario executed all but 1 operation on surrogate devices and took 1.48 hours to complete, which beat the historical algorithm's time of 1.79 hours but fell short of the experimental algorithm's time of 1.02 hours to complete. This algorithm's success at beating the historical algorithm using blind chance points to the benefits of cyber foraging and strengthens the support for additional research into using a lightweight heuristic to guide offloading sections. Although the random algorithm potentially chooses slower surrogates when faster surrogates were available, it shows that a minimal overhead algorithm can rival historical-based algorithms in certain scenarios.

A Performance Comparison of the Experiments

An analysis of the historical, experimental, and random algorithms' performance during the saturated scenario (Figure 28) shows that the overall, the historical and experimental algorithms share the same performance graph except during the cold-start

state where the historical algorithm is required to perform operations locally until training has been completed. Operations 1, 50, and 100 show the cold-start states and it can be clearly seen that the experimental algorithm can quickly change to the new operation and select a beneficial surrogate while the historical algorithm requires training. The random algorithm can be seen selecting surrogates at random within the pool of the 5 available surrogates.

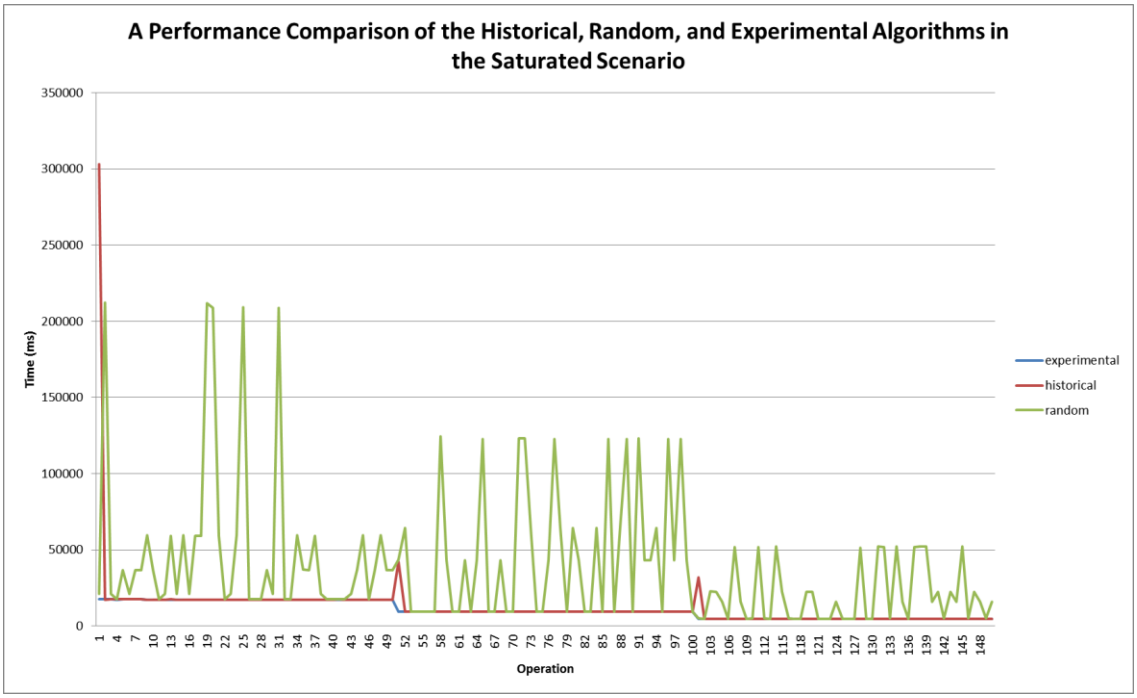

Figure 28 – A Performance Comparison of the Historical, Random, and Experimental Algorithms in the Saturated Scenario

An analysis of the historical, experimental, and random algorithms' performance during the slowly churning scenario shows that overall, the historical and experimental algorithms share the same performance graph (Figure 29) except during the cold-start state. The slowly churning scenario starts with no surrogates available and surrogates come online one at a time until all surrogates are available for use. Again, at operations 3, 53, and 103, the historical algorithm is required to perform operations locally until training has been completed while the experimental algorithm is quickly able to utilize

the newly arrived surrogate. The random algorithm, favoring remote execution also immediately utilizes the new surrogate by default since it is the only surrogate available. As seen previously, at operation 13, a new surrogate arrives and again, the experimental algorithm is able to recognize that this surrogate is more powerful and utilizes it immediately while the historical algorithm trains the surrogate. At this point, the random algorithm has 2 surrogates to choose from and oscillates between them. As more surrogates become available, both the historical and experimental algorithms do not change the selected surrogate while the random algorithm randomly selects surrogates from the slowly growing pool of available surrogates. This scenario repeats for the contrast and grayscale operations that start at operation 50 and 100, respectively.

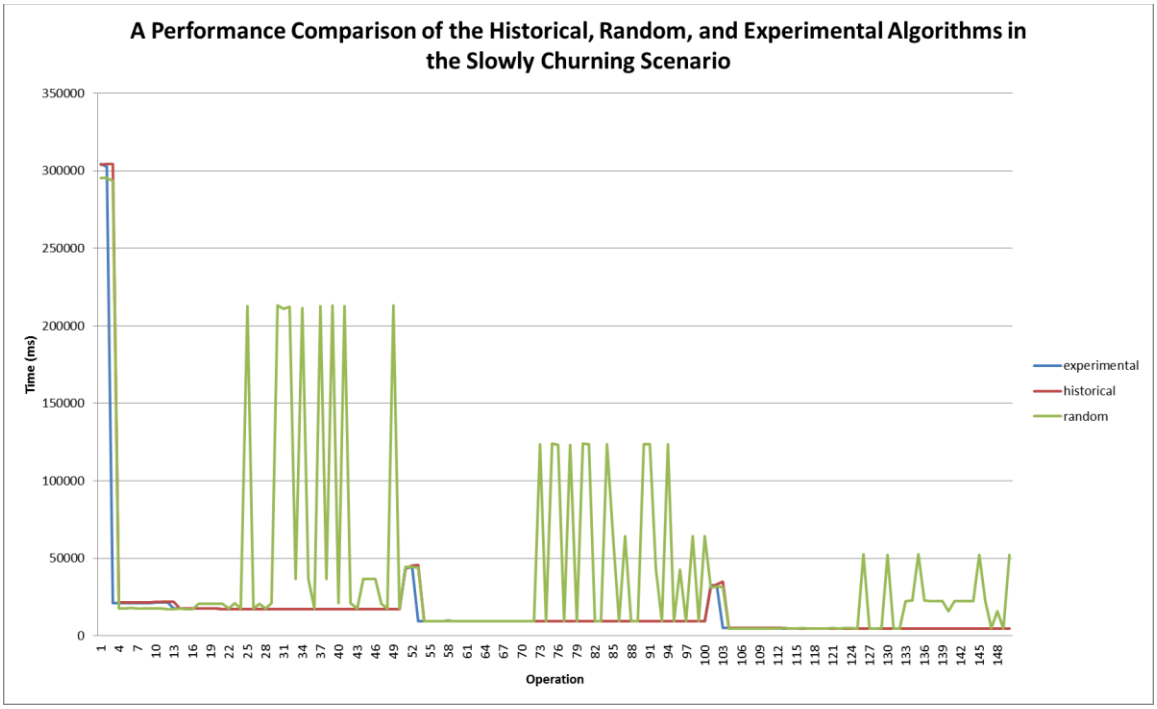

Figure 29 – A Performance Comparison of the Historical, Random, and Experimental Algorithms in the Slowly Churning Scenario

An analysis of the historical, experimental, and random algorithms' performance during the quickly churning scenario shows a sharp deviation (Figure 30) between the

historical algorithm and the experimental algorithm. In this scenario, surrogates arrive, leave the environment, and do not retain their training history between connections (to simulate every connection being a new device encounter). This is evident by the frequent spikes in the historical algorithm's performance compared to the experimental algorithm while the historical algorithm is training the newly arrived surrogates. In this scenario, the blind choosing of surrogates from a limited surrogate pool is often beneficial. This can be seen by the random algorithm closely following the experimental algorithm when only a few surrogates are available and the random chance picks a favorable surrogate.

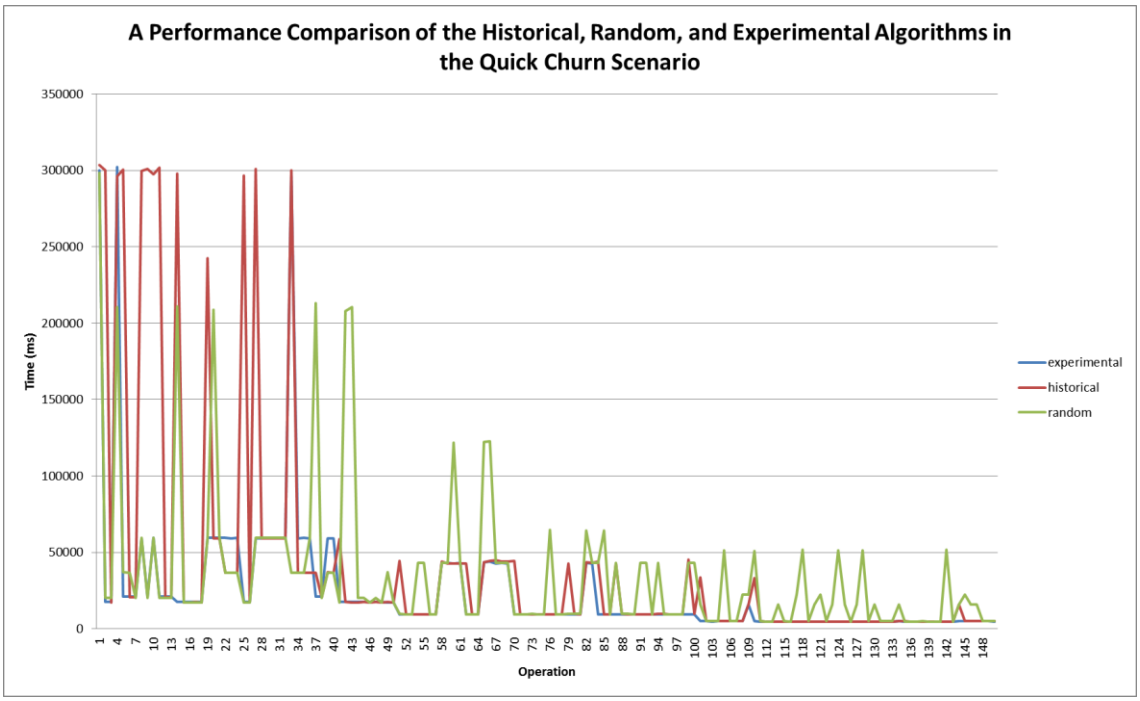

Figure 30 – A Performance Comparison of the Historical, Random, and Experimental Algorithms in the Quickly Churning Scenario

Scheduling Algorithm Overhead

The historical scheduling algorithm utilized simple linear regression to predict the execution time of operation based upon past observations. This required each surrogate to store data about each operation a surrogate could perform and required each connected surrogate to maintain a historical dataset for each operation. This scheduling algorithm

scaled O(n) and required the linear regression prediction method to be invoked for each node to determine the most beneficial surrogate for the current operation and image size. In addition to this overhead, jScavenger will request each new surrogate to perform the required operation on both a full-size image and a thumbnail image upon arrival. This additional training overhead occurs on a background thread. In the saturated scenario, the additional training overhead was 15 individual training sessions, the slowly churning scenario required 15 unique training sessions, and the quickly churning scenario required 64 training sessions.

The experimental scheduling algorithm utilized the surrogate speed rating and the approximate number of bytecode contained within an operation as a heuristic to choose a surrogate. The operational overhead for this algorithm, once the speed rating and the bytecode count was determined, was minimal. The surrogates were stored in a sorted list ranked by the anticipated speed of the operation. Retrieving a beneficial surrogate from the list was an O(1) operation.

The random scheduling algorithm generated a random number between 1 and the size of the currently connected surrogate list. The scheduler simply utilized the surrogate associated with the random number that was generated. This yielded an ultra-low overhead scheduling algorithm that could be implemented as O(1) if implemented in a data structure that supported direct access. In this research, the algorithm scaled O(n) because the surrogates were stored in a linked list which does not support direct access.

Summary of Results

Three experiments were conducted for this research, one for each of the scheduling algorithms used to select a surrogate to remotely execute image-processing operations. The scheduling algorithms consisted of a historical algorithm, an

experimental algorithm, and random algorithm. Each experiment consisted of the same 4 operational scenarios: disconnected, saturated, slowly churning, and quickly churning. Each operational scenario executed 300 operations consisting of sharpening an image, adjusting the contrast of an image, and converting an image to grayscale.

It was found that the historical scheduling algorithm required 8.25 hours to execute the 1,200 operations across all 4 operational scenarios with 93 individual training sessions. The experimental scheduling algorithm required 7.38 hours to complete all 1,200 operations across the 4 operational scenarios with no training required. The random scheduling algorithm required 10.2 hours to complete the 1,200 operations contained in the operational scenarios with no training overhead.

It was found that the historical and experimental algorithms performed consistently when the environment was static or slowly changing differing only by the number of training sessions required. When the frequency of change increased, the performance of the historical and experimental algorithms quickly diverged. This was because the training requirement of the historical algorithm prevented the use of newly arrived surrogates until training had been completed. The experimental algorithm was shown to be able utilize beneficial surrogates immediately, without training, which allotted for the performance gain when the rate of change increased.

As expected, the random scheduler was the slowest of the three algorithms with one exception. The random algorithm achieved a faster execution time for the quickly churning scenario than the historical algorithm did for the same scenario. The random scheduling algorithm beat the historical algorithm by 31 minutes. This reinforces the

benefits of cyber foraging and suggests that even unguided offloading in an unknown environment may be beneficial.

Overall, the experimental scheduling algorithm presented in this research outperformed both the historical and random scheduling algorithms for all scenarios (excluding the disconnected scenario). This achieved the goal of this research of increasing the performance of cyber foraging application by decreasing the overall execution time of a cyber foraging application.

Chapter 5

Conclusions, Implications, Recommendations, and Summary

Conclusions

This research has as shown that a heuristic-based scheduling algorithm can increase the performance of a cyber foraging application by decreasing the application's runtime. The success of the experimental algorithm was attributed to the algorithm's ability to utilize beneficial surrogates faster, rather than delaying remote execution while the surrogate was trained.

Three experiments were conducted as a part of this research. The first experiment investigated the use of a historical scheduling algorithm to offload operations for remote execution. The second experiment utilized the experimental scheduling algorithm, which utilized a heuristic-based scheduling algorithm to offload operations for remote execution. The third experiment utilized a random scheduling algorithm that randomly selected a surrogate from the list of available surrogates for use. The results of each experiment were then compared to derive the performance analysis.

The results of the historical experiment showed that the training overhead was directly related to the overall surrogate-churn. During the saturated test, 15 individual training sessions were required. Since all five surrogates were available for the duration of the test, 5 training sessions were required for each of the three operations conducted during the test. The slowly churning test also required 15 training sessions, as each of the surrogates were utilized during the duration of the test. The quickly churning scenario saw the greatest training overhead as 64 individual training sessions were required because surrogates frequently joined and left the network.

The random scheduler experiment showed that blindly offloading operations to surrogates could be beneficial in scenarios where there are relatively few surrogates. In the saturated scenario, where all 5 surrogates were available, the random scheduler produced the longest execution time of all the scheduling algorithms. This can be attributed to the scheduler picking surrogates that were better than local execution, but far slower than the most beneficial surrogate available. On the other end of the spectrum, in the quickly churning scenario, where there were relatively few surrogates to choose from, the random selection of surrogates produced a faster execution time that the historical scheduling algorithm. This can be attributed to the high probability of selecting a beneficial surrogate from the small pool of available surrogates.

The results of the experimental experiment showed that a heuristic-based scheduling algorithm was able to decrease the execution time of an application. The experimental scheduling algorithm achieved the fastest overall application execution times of all the scheduling algorithms, beating the historical algorithm by 0.87 of an hour (10.5%) and the random algorithm by 2.82 hours (28%).

Implications

This research provided an approach to surrogate selection during the cold-start state that fast tracks the utilization of new surrogates or new operations on existing surrogates when there was a lack of past performance data. The experimental algorithm has demonstrated the ability to eliminate the cold-start state in historically based scheduling algorithms. This suggests that the experimental algorithm may be implemented without the need of a historical component, thus eliminating the overhead of storing performance data and performing prediction calculations.

Recommendations

The goal of the research was met by the experimental algorithm; however, there are some areas of research could be pursued. Kristensen (2010) utilized the NBench2 benchmark suite to produce surrogate strengths, but the benchmark was a heavyweight process and required about 10 minutes to execute per device. The heuristic utilized in this research was lightweight and reduced the overhead required by leveraging the execution of the software itself to generate the data. Additional research into determining the minimum amount of data required to create a useful heuristic for use in the offloading of operations could further reduce the effort required to generate the heuristics.

The differences between CPU architecture and instruction execution speed could also be investigated to determine the architectural impact native code execution has upon the heuristic. Kristensen and Bouvin (2010) observed that the different CPU architectures influenced the operation weights in their heuristic. The operation weights for the PowerPC CPU architecture were almost three times as high when compared to the Intel architecture for the same operation. Additional research into how the architectural difference can be incorporated into a heuristic will allow for a smoother application of the heuristic in a heterogeneous environment.

This research focused heavily upon CPU performance; however, the incorporation of other subsystems could provide a more accurate heuristic. The performance of Hadoop nodes in a cloud computing environment incorporated the use of disk I/O performance, memory performance, and network performance in the overall evaluation of node performance (Lin & Liu, 2013). The authors' state that this was required due to the diversity of individual nodes in the Hadoop cluster diverging over time, due to hardware

failures and upgrades. The incorporation of additional heuristics could improve the overall performance estimate, especially if the operations rely heavily upon a subsystem that is not currently accounted for in the general heuristic.

In an effort to optimize the scheduling of tasks in a cloud environment, the task requirements and server capabilities are required so that tasks can be pared with the most suitable server (Gupta, Fritz, Price, Hoover, De Kleer, & Witteveen, 2013). Gupta, et al. (2013) utilized offline training to build a historical dataset of server and job performance for use in scheduling because no method currently exists to estimate server performance from hardware specifications. The use of a Bayesian estimator to produce a performance heuristic for each surrogate based upon hardware specifications could potentially provide performance heuristics for use in task scheduling.

Summary

Mobile devices due to their size, weight, and power constraints typically lag behind stationary desktop workstations where processing power, memory, and storage capacity are concerned. The cyber foraging paradigm enables mobile devices to perform beyond their means by offloading code for remote execution. By remotely executing code, an application can conserve memory and battery power by allowing surrogate machines to expend the resources rather than requiring the mobile device itself to expend the precious resources. The remote execution of code may also allow for the overall execution time of the process to be shortened or the fidelity of the result to be increased due to the utilization of high-performance computers rather than the resource poor mobile device.

A barrier to making offloading decisions in a cyber foraging system centers on obtaining enough information to make informed remote execution decisions. Given ample time and processing power, an execution scheduler can enumerate all available surrogates to determine the optimum surrogate to utilize in a given situation; however, in a highly interactive environment, the time required to make such a determination may be greater than what the end-user is willing to accept. The price may also be higher in terms of the processing power and the battery power expended to make the offloading decision than would be gained by remotely executing the operation. Compounding this issue is the cold-start problem, which potentially delays the availability of a newly arrived surrogate because the system does not have enough information available to rank the surrogate for remote execution scheduling.

This research achieved the goal of utilizing metrics obtained from the run-time profiling of a Java program to decrease the run-time of a cyber foraging application. This was accomplished by scheduling beneficial offloading decisions during the cold-start state. The utilization of run-time metrics from the applications themselves provided a heuristic that does not require a-priori training, design-time information from the developer, or training effort from the end-user in order for the system to make informed offloading decisions.

The methodology utilized to obtain the metrics for the heuristic was based upon the speed that a surrogate demonstrated it could execute Java bytecode instructions and the average number of instructions contained within an operation. To obtain the speed rating for each surrogate, the surrogate client application was profiled to obtain the average number of instructions that were expected to be executed during the initialization

of the client. The operations were profiled by generating a control flow graph for the operation and calculating the average number of instruction that could potentially be executed by the operation. The surrogate selection process calculates the potential execution speed for the operation by dividing the expected number of instructions in the operation by the speed of the surrogate.

Three experiments were conducted, one for each of the scheduling algorithms used to select a surrogate for remote execution. The scheduling algorithms consisted of a historical algorithm, an experimental algorithm, and random algorithm. Each experiment consisted of the same 4 operational scenarios: disconnected, saturated, slowly churning, and quickly churning.

The disconnected operation scenario tested the system in an environment where there are no surrogates available. The saturated environment scenario tested the system in an over-provisioned environment where all 5 surrogates were available. The slowly churning environment scenario exercised the system in an environment where there were initially no surrogates and surrogates were slowly added to the system until all five surrogates were available for use. The quickly churning environment scenario exercised the system in an environment where initially no surrogates were available. Surrogates were then simultaneously added and removed from the system after every few operations. Each operational scenario executed 300 operations consisting of sharpening an image, adjusting the contrast of an image, and converting an image to grayscale.

It was found that the historical scheduling algorithm required 8.25 hours to execute the 1,200 operations across all 4 operational scenarios with 93 individual training sessions. The experimental scheduling algorithm required 7.38 hours to complete all

1,200 operations across the 4 operational scenarios with no training required. The random scheduling algorithm required 10.2 hours to complete the 1,200 operations contained in the operational scenarios with no training overhead.

Overall, the experimental scheduling algorithm presented in this research outperformed the historical scheduling algorithm by 10.5% and the random scheduling algorithm by 28%. This achieved the research goal by decreasing the overall execution time of a cyber foraging application.

Appendices

Appendix A: Sample Java Program

The sample Java program used in generating the sample CFG.

```
1. public class SimpleJavaProgram
2. {
3.     public static void main(String[] args)
4.     {
5.           int a = 5;
6.           int b = 10;
7.
8.           System.out.println("A = " + a);
9.           System.out.println("B = " + b);
10.
11.          if( a > 5 )
12.          {
13.                a++;
14.                b--;
15.          }
16.          else
17.          {
18.                a--;
19.                b++;
20.          }
21.
22.          System.out.println("A = " + a);
23.          System.out.println("B = " + b);
24.    }
25.}
```

Appendix B: Sample Java Program – Bytecode Representation

Command line used to generate and capture the output was:

javap -v SimpleJavaProgram.class > output.txt

The Java version used to generate this output: java version "1.7.0_17"

Note: This output has been altered. The comments have been removed from the output due to space and formatting considerations.

```
--Start Listing--
         0: iconst_5
         1: istore_1
         2: bipush        10
         4: istore_2
         5: getstatic     #16
         8: new           #22
        11: dup
        12: ldc           #24
        14: invokespecial #26
        17: iload_1
        18: invokevirtual #29
        21: invokevirtual #33
        24: invokevirtual #37
        27: getstatic     #16
        30: new           #22
        33: dup
        34: ldc           #42
        36: invokespecial #26
        39: iload_2
        40: invokevirtual #29
        43: invokevirtual #33
        46: invokevirtual #37
        49: iload_1
        50: iconst_5
        51: if_icmple     63
        54: iinc          1, 1
        57: iinc          2, -1
        60: goto          69
        63: iinc          1, -1
        66: iinc          2, 1
        69: getstatic     #16
        72: new           #22
        75: dup
        76: ldc           #24
        78: invokespecial #26
        81: iload_1
        82: invokevirtual #29
        85: invokevirtual #33
        88: invokevirtual #37
        91: getstatic     #16
        94: new           #22
```

```
         97: dup
         98: ldc           #42
        100: invokespecial #26
        103: iload_2
        104: invokevirtual #29
        107: invokevirtual #33
        110: invokevirtual #37
        113: return
--End Listing--
```

Appendix C: Sample Execution Log File Data

Below is an excerpt from the execution log. The header row and empty lines between entries have been added to increase readability.

```
sequence number~record type~record data

1~1<request><requestID>1</requestID><requestType>RPC</requestType><operationNa
me>ImageLib.Contrast</operationName><fileName>dog.png</fileName><parameters><p
arameter><parameterName>imageBytes</parameterName><parameterDirection>IN</para
meterDirection><parameterDataType>byte[]</parameterDataType><parameterValue></
parameterValue></parameter></parameters></request>

2~2~<response><responseID>1</responseID><operationName>ImageLib.Contrast</oper
ationName><executionTime>32421</executionTime><errorCode>0</errorCode><errorDe
scription></errorDescription><parameters><parameter><parameterName>RETURN_VALU
E</parameterName><parameterDirection>OUT</parameterDirection><parameterDataTyp
e>byte[]</parameterDataType><parameterValue>X62IBNhchbxBwbGhVwc==</parameterVa
lue></parameter></parameters>

3~3~ImageLib.Contrast~dog.png~6209174~surrogate-2~785~421~surrogate-
1;surrogate-2;surrogate-3;surrogate-4;surrogate-5~cold-
start;online;offline;online;online

4~1~<request><request><requestID>2</requestID><requestType>RPC</requestType><o
perationName>ImageLib.Contrast</operationName><fileName>dog.thumb.png</fileNam
e><parameters><parameter><parameterName>imageBytes</parameterName><parameterDi
rection>IN</parameterDirection><parameterDataType>byte[]</parameterDataType><p
arameterValue></parameterValue></parameter></parameters></request>

5~2~<response><responseID>2</responseID><operationName>ImageLib.Contrast</oper
ationName><executionTime>421</executionTime><errorCode>0</errorCode><errorDesc
ription></errorDescription><parameters><parameter><parameterName>RETURN_VALUE<
/parameterName><parameterDirection>OUT</parameterDirection><parameterDataType>
byte[]</parameterDataType><parameterValue>X62IBNhchbxBwbGhVwc==</parameterValu
e></parameter></parameters>

6~3~ImageLib.Contrast~dog.png~7194~surrogate-2~285~121~surrogate-1;surrogate-
2;surrogate-3;surrogate-4;surrogate-5~cold-start;online;offline;online;online

-end-
```

Appendix D: Sample Driver File – Saturated Scenario

```
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
```

```
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
```

Appendix E: Sample Driver File – Slowly Churning Scenario

```
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
```

```
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
```

```
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
```

```
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
```

```
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
```

```
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
```

Appendix F: Sample Driver File – Quickly Churning Scenario

```
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
```

```
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC SHARPEN s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC SHARPEN s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
```

```
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
```

```
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC CONTRAST s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=OFFLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
```

```
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
```

```
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=OFFLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=ONLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=ONLINE,s-2=OFFLINE,s-3=OFFLINE,s-4=ONLINE,s-5=OFFLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
RPC GRAYSCALE s-1=OFFLINE,s-2=ONLINE,s-3=OFFLINE,s-4=ONLINE,s-5=ONLINE;
```

References

Albert, E., Arenas, P., Genaim, S., Puebla, G., & Zanardini, D. (2007). Cost analysis of java bytecode *Programming Languages and Systems* (pp. 157-172): Springer.

Balan, Flinn, J., Satyanarayanan, M., Sinnamohideen, S., & Yang, H. I. (2002). *The case for cyber foraging.* Paper presented at the ACM SIGOPS European Workshop.

Balan, Gergle, D., Satyanarayanan, M., & Herbsleb, J. (2007). *Simplifying cyber foraging for mobile devices.* Paper presented at the Proceedings of the 5th international conference on Mobile systems, applications and services.

Balan, Satyanarayanan, M., Park, S. Y., & Okoshi, T. (2003). *Tactics-based remote execution for mobile computing.* Paper presented at the Proceedings of the 1st international conference on Mobile systems, applications and services.

Binder, W., & Hulaas, J. (2006). Using bytecode instruction counting as portable CPU consumption metric. *Electronic Notes in Theoretical Computer Science, 153*(2), 57-77.

Chun, B. G., Ihm, S., Maniatis, P., & Naik, M. (2010). Clonecloud: boosting mobile device applications through cloud clone execution. *arXiv preprint arXiv:1009.3088*.

Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R., et al. (2010). *MAUI: making smartphones last longer with code offload.* Paper presented at the Proceedings of the 8th international conference on Mobile systems, applications, and services.

Flinn, J., Park, S. Y., & Satyanarayanan, M. (2002). *Balancing performance, energy, and quality in pervasive computing*.

Flinn, J., & Satyanarayanan, M. (1999). Energy-aware adaptation for mobile applications. *ACM SIGOPS Operating Systems Review, 33*(5), 48-63.

Flynn, J. (2012). *Cyber Foraging: Bridging Mobile and Cloud Computing*: Morgan & Claypool.

Gu, X., Nahrstedt, K., Messer, A., Greenberg, I., & Milojicic, D. (2003). *Adaptive offloading inference for delivering applications in pervasive computing environments.* Paper presented at the Pervasive Computing and Communications, 2003.(PerCom 2003). Proceedings of the First IEEE International Conference on.

Gu, X., Nahrstedt, K., Messer, A., Greenberg, I., & Milojicic, D. (2004). Adaptive offloading for pervasive computing. *Pervasive Computing, IEEE*, *3*(3), 66-73.

Gupta, S., Fritz, C., Price, B., Hoover, R., De Kleer, J., & Witteveen, C. (2013). *ThroughputScheduler: Learning to Schedule on Heterogeneous Hadoop Clusters.* Paper presented at the ICAC.

Gurun, S., Krintz, C., & Wolski, R. (2004). *NWSLite: a light-weight prediction utility for mobile devices.* Paper presented at the Proceedings of the 2nd international conference on Mobile systems, applications, and services.

Huerta-Canepa, G., & Lee, D. (2008). *An adaptable application offloading scheme based on application behavior.* Paper presented at the Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on.

Irwin, J., Kickzales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., et al. (1997). Aspect-oriented programming. *Proceedings of ECOOP, IEEE, Finland*, 220-242.

Kafaie, S., Kashefi, O., & Sharifi, M. (2011). *Augmented Mobile Devices through Cyber Foraging.* Paper presented at the 10th International Symposium on Parallel and Distributed Computing, Cluj Romania.

Kemp, Palmer, N., Kielmann, T., Seinstra, F., Drost, N., Maassen, J., et al. (2009). *eyeDentify: Multimedia cyber foraging from a smartphone.* Paper presented at the Multimedia, 2009. ISM'09. 11th IEEE International Symposium on.

Kemp, R., Palmer, N., Kielmann, T., & Bal, H. (2012). Cuckoo: a computation offloading framework for smartphones. *Mobile Computing, Applications, and Services*, 59-79.

Kristensen, M. D. (2010). *Scavenger: Transparent development of efficient cyber foraging applications.* Paper presented at the Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on.

Kristensen, M. D., & Bouvin, N. O. (2010). Scheduling and development support in the scavenger cyber foraging system. *Pervasive and Mobile Computing, 6*(6), 677-692.

Liagouris, J., Athanasiou, S., Efentakis, A., Pfennigschmidt, S., Pfoser, D., Tsigka, E., et al. (2011). Mobile task computing: beyond location-based services and ebooks. *Web and Wireless Geographical Information Systems*, 124-141.

Lin, W., & Liu, J. (2013). Performance Analysis of MapReduce Program in Heterogeneous Cloud Computing. *Journal of Networks, 8*(8), 1734-1741.

Narayanan, D., Flinn, J., & Satyanarayanan, M. (2000). *Using history to improve mobile application adaptation.* Paper presented at the Mobile Computing Systems and Applications, 2000 Third IEEE Workshop on.

Narayanan, D., & Satyanarayanan, M. (2003). *Predictive resource management for wearable computing.* Paper presented at the Proceedings of the 1st international conference on Mobile systems, applications and services.

Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J., & Walker, K. R. (1997). *Agile application-aware adaptation for mobility*.

Pooranian, Z., Shojafar, M., Abawajy, J. H., & Singhal, M. (2013). Gloa: a new job scheduling algorithm for grid computing. *IJIMAI, 2*(1), 59-64.

Satyanarayanan, M. (1996). *Fundamental challenges in mobile computing.* Paper presented at the Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing.

Satyanarayanan, M. (2001). Pervasive computing: Vision and challenges. *Personal Communications, IEEE, 8*(4), 10-17.

Schein, A. I., Popescul, A., Ungar, L. H., & Pennock, D. M. (2002). *Methods and metrics for cold-start recommendations.* Paper presented at the Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval.

Serral, E., Valderas, P., & Pelechano, V. (2011). Improving the Cold-Start Problem in User Task Automation by Using Models at Runtime. *Information Systems Development*, 671-683.

Sharifi, M., Kafaie, S., & Kashefi, O. (2011). A survey and taxonomy of cyber foraging of mobile devices.

Verbelen, T., Simoens, P., De Turck, F., & Dhoedt, B. (2011). *AIOLOS: mobile middleware for adaptive offloading*. Paper presented at the Proceedings of the Workshop on Posters and Demos Track.

Verbelen, T., Simoens, P., De Turck, F., & Dhoedt, B. (2012). AIOLOS: middleware for improving mobile application performance through cyber foraging. *Journal of Systems and Software*.

Wang, Y., & Li, F. (2009). Vehicular ad hoc networks *Guide to wireless ad hoc networks* (pp. 503-525): Springer.

Zhang, X., Kunjithapatham, A., Jeong, S., & Gibbs, S. (2011). Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. *Mobile Networks and Applications, 16*(3), 270-284.