

Transparent Optimization of Grid Server Selection With Real-Time Passive Network Measurements

Marcia Zangrilli and Bruce B. Lowekamp
Computer Science Department
College of William and Mary
Williamsburg VA, USA
{mazang,lowekamp}@cs.wm.edu

Abstract

Grid services have tremendously simplified the programming challenges in leveraging large-scale distributed computing. At the same time, the increased level of abstraction reduces the opportunities available to the application for optimizing its performance by monitoring the system. In this paper we introduce a monitoring grid services proxy, which transparently monitors network performance and selects between several replica service providers. This approach provides optimized server selection without any modification to or even awareness of the client application or service providers. We describe how we implement the proxy and monitor the available bandwidth to the service providers using the Wren monitoring toolkit. We present analysis indicating that our monitoring has negligible overhead. Finally, we demonstrate the practicality of our approach by optimizing the server selection for INCOGEN's VIBE, a bioinformatics workflow application that uploads gene sequences for analysis by remote service providers.

1. Introduction

One of the challenges in developing grid applications is the difficulty in creating applications that can function across both high latency networks and tightly coupled clusters. Grid services have emerged as a means to help the coordination of grid applications by providing a standard interface between clients and services. Grid services help reduce the programming complexities for clients and create opportunities to help steer the decision of which service is best for the client to invoke.

In this paper, we explore the use of a grid service aware proxy server to provide transparent optimization services to grid services applications. The proxy service offers a new level of abstraction that hides the exact data or service resource from the client. In such a system, the client is configured with the location of the proxy as the address for all ser-

vices, but it is totally unaware of the behavior of the proxy and all optimization decisions are transparent to both the service providers and the client.

While adding proxies to grid services architectures may remove some control from the client, our goal is to provide the middleware with sufficient power to make the appropriate performance optimizations without the need to involve either the user or the client application unless qualitative decisions—such as choosing between databases managed by different groups—need to be made.

Network performance monitoring is critical for running applications in complex grid environments, but obtaining this information often requires the proxy to actively probe the network for available bandwidth. The active approach produces accurate measurements, but it may cause competition between application traffic and the measurement traffic, reducing the performance of other applications. Most of these active algorithms rely on UDP traffic to probe the path for available bandwidth, whereas most grid applications use TCP traffic. If UDP traffic is packet-shaped differently than TCP traffic, measurements made using UDP traffic may not reflect the actual bandwidth available to TCP applications. For available bandwidth measurements to be useful to grid services proxies, they must be both accurate and non-invasive.

Our solution is to incorporate passive Wren measurements into a grid services proxy to help facilitate the proxy's decision of which service to invoke. Wren is less invasive than active probing because it uses passive traces of the traffic already flowing through the proxy to provide available bandwidth measurements. Because Wren measures available bandwidth using the application's own traffic, the measurements provided by Wren will reflect the bandwidth available to that application.

Wren will allow the proxy server to choose between different options for executing the same service, e.g., whether to process data locally on a single server or to spend the time uploading the dataset to a remote high-performance cluster that offers the same service. These features will be used to hide the complex details of service and data loca-

tion from the client application, while still exposing options of different services and datasets to the client and the user. Furthermore, Wren measurements enable the proxy to provide feedback to the client on how long each service module will take to transfer data or client requests.

The principle contribution of our work is the implementation and analysis of a system that uses real-time measurements to make runtime server selection decisions without modification of either the client application or service provider's code or infrastructure.

In this paper we describe the online Wren system and the integration of Wren with a simple grid services proxy. We have previously described the preliminary design and implementation of Wren [15, 16]. This paper significantly extends our previous work by providing:

- Introduction and analysis of the online Wren system, completing the circle by providing real-time measurements back to the application for performance optimization,
- Application of passive monitoring to a server selection proxy that functions in a grid services environment without modification of any other component,
- Experimental results that demonstrate the accuracy of Wren measurements using grid service requests,
- Quantification of lag associated with providing real-time Wren measurements, and
- Demonstration that Wren measurements can aid the proxy in server selection.

In Section 2 we review related work on grid services and define available bandwidth. Section 3 describes the implementation of the Wren monitoring system and shows the efficiency of Wren tracing. Section 4 analyzes the overheads of tracing proxy traffic, the accuracy of measuring available bandwidth, and the measurement lag associated with Wren measurements. Finally, Section 5 describes how Wren measurements can be used by a grid service aware proxy to improve client performance.

2. Background

2.1. Grid Services

As an alternative to the complex challenge of programming distributed grid resources, the area of grid services has gained ground as a means of providing researchers with programmatic access to distributed computational resources [1, 14]. In many fields the same applications are used repeatedly (e.g. BLAST in genomics or charm++ in biochemistry). Grid services that execute a known application on

user-submitted data provide users with powerful resources and help reduce the programming complexity of developing grid applications.

Grid services are at the core of the effort to provide seamless interoperability among resources in distributed systems. Grid services are fundamentally Web Services that provide specific functionality. The OGSI defines Grid Services with extensions to the standard Web Service Definition Language (WSDL) to provide: sophisticated security infrastructure; standard service invocation mechanism for service lifetime management; state management and modeling of time; and standard service inquiry mechanism [14].

The OGSA Execution Management Services (OGSA-EMS) [1] are a set of components that are collectively responsible for identifying service locations, selecting the most appropriate location, and running the service. EMS itself comprises multiple services that achieve these goals and are designed in a composable fashion such that services can be connected as required. They leverage information provided through a Grid Monitoring Architecture to provide their services. In our case, the services provided by our proxy are most similar to an OGSA Job Manager. By combining the Wren monitoring techniques with a Message Broker that is used to deliver messages between components, the same level of transparent, passive network performance monitoring could be achieved, and that information could be used to make resource mapping decisions in combination with other information available. However, as our current applications are not based on an OGSA-compliant library, we have not yet pursued this option.

2.2. Available Bandwidth

Available bandwidth describes what portion of the path is currently unused by other competing traffic. More precisely, available bandwidth is determined by subtracting the utilization from the capacity of the network path [7, 11]. Available bandwidth is useful for applications that are considering increasing their utilization, for a new application approximating what bandwidth might be available, and for network engineers monitoring the performance of their network.

One technique for measuring available bandwidth is the self-induced congestion (SIC) technique. The basic principle of SIC is that if packets are sent at a rate larger than the available bandwidth, the queuing delays will have an increasing trend, and the rate the packets arrive at the receiver will be less than the sending rate. If the one-way delays are not increasing and the rate the packets arrive is the same as the sending rate of the packets, then the available bandwidth is at least equal to the sending rate. Tools that utilize this concept actively probe the network path for the largest sending rate that does not result in queuing delays with an

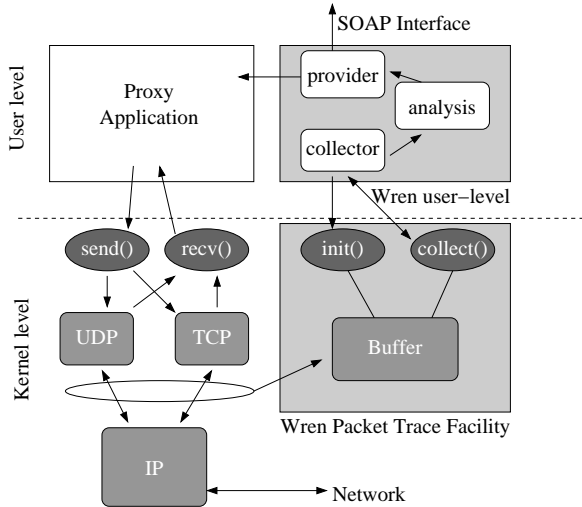


Figure 1. Integration of Wren packet trace facility and Wren analysis to a grid services proxy provides transparent monitoring of network resources.

increasing trend because this sending rate reflects the available bandwidth of the path.

Tools that use principles similar to self-induced congestion include Netest [6], PathChirp [12], Pathload [5], PTR [3], and TOPP [10]. Unlike these tools, all of which actively inject traffic into the network, Wren passively applies the self-induced congestion principle to existing application traffic to measure available bandwidth.

A similar project to Wren is Inline measurement TCP (ImTCP). ImTCP integrates an active self-induced congestion algorithm into a TCP stack, waiting until the congestion window has opened large enough to send an appropriate length train and then delaying packet transmissions until enough packets are queued to generate a precisely spaced train [8]. Wren avoids modifying the TCP sending algorithm and, in particular, avoids delaying packet transmission.

3. Online Wren Monitoring System

The Wren bandwidth monitoring system [2, 15, 16] provides available bandwidth measurements by passively observing existing application traffic. Figure 1 shows an overview of the Wren architecture. Wren is separated into a kernel-level packet trace component and a user-level processing component. The kernel-level packet trace facility is responsible for gathering information associated with incoming and outgoing packets and the user-level is used to collect the traces from the kernel. Run-time analysis de-

termines available bandwidth and the measurements are reported to other applications through a SOAP interface. Alternatively, the packet traces can be transmitted to a remote repository.

In the remainder of this section we will describe the kernel-level Wren packet trace facility, the efficiency of Wren tracing, and the processing done by the Wren user-level component, with an emphasis on the Wren available bandwidth algorithm.

3.1. Wren Packet Trace Facility

We use the Wren packet trace facility to collect traces of TCP traffic sent between the grid services proxy and the servers accessed by the clients. Wren timestamps packets in the kernel as they arrive or leave, collects other protocol specific information, and passes the trace to the user-level for analysis. Wren can capture full traces of small bursts of traffic or periodically capture smaller portions of continuous traffic and use those to measure available bandwidth. For high bandwidth-delay product networks, the tool can also ensure that it collects data transmission and ACK receipt logs of the same range of sequence numbers.

The key benefit of the Wren packet trace facility is its use of kernel-level timestamps. Wren timestamps outgoing packets in the kernel immediately before they are handed to the layer-2 device driver and similarly timestamps incoming packets as soon as they are passed to the TCP code. The precision of the timestamps is crucial because our passive available bandwidth algorithm relies on observing the behavior of small groups of packets on the network. This additional accuracy, combined with the efficiency of collecting only the information needed while reading from the buffer at regular intervals, provides the traces required for accurate monitoring without disturbing the running application.

3.1.1 Wren Tracing Efficiency

The principle alternative to a system like the Wren Packet Trace Facility is using the Berkeley packet filter [9] to capture TCP traffic at the kernel-level. The most straightforward approach is to use *tcpdump* to capture packet header traces and collect the data from the timestamps and packet headers. We have implemented this approach, as well as a direct *libpcap* implementation that extracts only the needed fields for later analysis.

We present the results of experimenting with packet tracing techniques between two 2.8GHz Hyperthreaded P4s running Linux 2.4.29 with 1GB of RAM and an Intel CSA (non-PCI) gigabit Ethernet NIC. The nodes were connected using a Cisco 3750 gigabit switch. Here we analyze only the cost of capturing packets and saving them to the hard-drive for offline analysis.

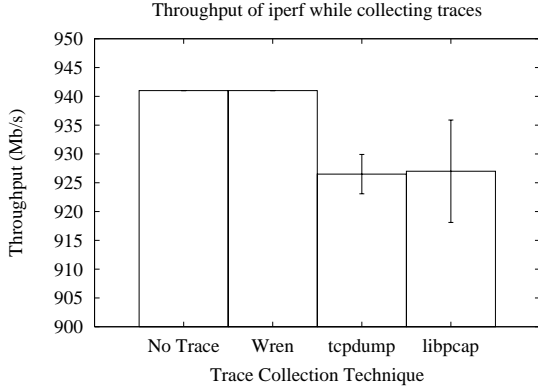


Figure 2. Throughput achieved by iperf with no packet trace collection, using Wren, using tcpdump, and using libpcap

At 100Mbps speeds, all three approaches can collect 100% of traffic at full speed with minimal affect on application performance. Therefore, we focus our efficiency analysis on Gbps traffic. Figure 2 presents the throughput achieved by iperf while using each of the three capture techniques, as well as without any packet capture. The Wren approach achieves the same throughput as the untraced connection; however the two packet-filter approaches both reduce throughput.

Most applications other than bulk data transfer perform both computation and communication. To simulate this behavior, we repeated the same experiment while also running a simple 1000×1000 dense matrix multiply. Figure 3 presents the execution time of the matrix multiply and Figure 4 presents the throughput achieved by iperf during the matrix multiplication. Again, the Wren packet trace facility clearly is the lowest overhead of the trace collection tools.

3.2. Wren User-Level Analysis

Figure 1 shows that the Wren user-level comprises the collector, analysis, and provider threads. The Wren Analysis thread processes traces acquired from the collector thread by parsing the traces into separate connections and then applying Wren’s passive available bandwidth algorithm to packets obtained for each connection. The parsing step also merges information about incoming and outgoing packets to form a complete packet picture for the Wren algorithm. The analysis thread passes Wren measurements to the provider thread that makes the measurements available to all interested applications through a SOAP interface.

The online Wren algorithm groups outgoing packets into trains by identifying sequences of packets with similar interdeparture times between successive pairs. The tool searches

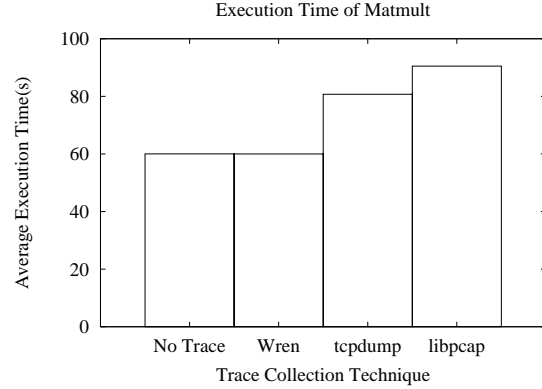


Figure 3. Execution time of a 1000×1000 matrix multiply while sending data using iperf with no packet trace collection, using Wren, using tcpdump, and using libpcap

for maximal-length trains with consistently spaced packets and calculates the initial sending rate (ISR) for those trains. After identifying a train, we calculate the ACK return rate for the matching ACKs. The available bandwidth is determined by observing the ISR at which the ACKs show an increasing trend in the RTTs, indicating congestion on the path.

The first step in our one-sided algorithm is to group packets into trains. We look at the relationship between the interdeparture times of sequential data packets. If interdeparture times of successive pairs are similar, then the packets are departing the machine at approximately the same rate. Let Δ_0 be the interdeparture time between data packets 0 and 1. To form a train, the interdeparture times Δ_i between each successive pair of packets i and $i+1$ in the train must satisfy the requirement that $\min_i(\log(\Delta_i)) > \max_j(\log(\Delta_j)) - \alpha$, essentially requiring consistent spacing between the packets. For these experiments, we accepted trains where $\alpha = 1$. Because of the bursty transmission of packets within any TCP flow [13], interdeparture times typically vary by several orders of magnitude even during bulk data transfers, therefore this approach selects only the more consistently spaced bursts as valid trains. We impose a minimum length of 7 packets for valid trains and select maximal length trains subject to the grouping requirement above.

Once a train is formed, the algorithm enters the processing phase. For each train that is formed, we calculate the the initial sending rate (ISR) by dividing the total number of bits in the train by the difference between the end time and the start time. The start time of the train refers to the time the first data packet in the train departs the machine, and the end time of the train specifies the time that the last data packet in the train departs the machine. The ISR of

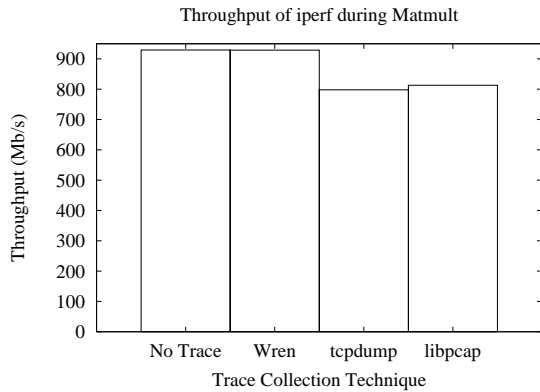


Figure 4. Throughput of iperf while running a 1000×1000 matrix multiply with no packet trace collection, using Wren, using tcpdump, and using libpcap

each train is compared to the ACK return rate. The ACK return rate is calculated by dividing the total number of bits in the train by the difference between the time the first ACK for the train and the last ACK for the train arrive back at the sending machine. Our algorithm only uses packets that have an explicit ACK packet, which is typically every other packet with delayed ACKs, although ACKs may be less frequent on high-bandwidth paths.

The next step in processing a train is to use a pairwise comparison test to determine the trend in the RTTs of the packets in that train. If $\forall i : RTT_i < RTT_{i+1}$ then the train has an increasing trend, indicating its ISR was higher than the available bandwidth. Otherwise, the train trend is labeled as non-increasing. If the train has a non-increasing trend and the ISR is similar in value to the ACK return rate, we know that the train ISR did not cause queuing on the path. Therefore, we report the ISR as a lower-bound available bandwidth measurement.

There are two sources of variation in Wren’s available bandwidth measurements: variation in the network’s available bandwidth and variation in the sending rate of the TCP flow. When multiple non-increasing trains are observed at different rates over very short time scales, using the highest ISR non-increasing train as the only measurement may be appropriate. However, as available bandwidth can change rapidly, distinguishing between TCP sender variability and network variability becomes more difficult; therefore the Wren tool provides a series of lower-bounds, with the intention that further statistical analysis is required according to the desired purpose of the data.

All available bandwidth observations are passed to the Wren provider thread. The provider thread provides a SOAP interface that clients can use to receive the stream

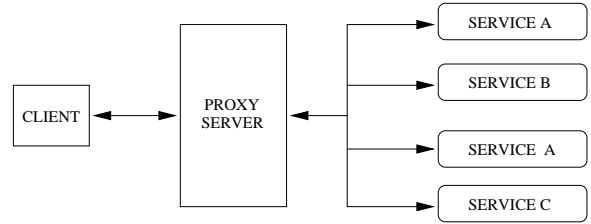


Figure 5. A grid services proxy provides a client with a single interface to multiple services.

of measurements produced using application traffic. Because the trains are short and represent only a singleton observation of an inherently bursty process, multiple observations are required to converge to an accurate measurement of available bandwidth.

4. Grid Services Proxy

As shown in Figure 5, a grid service proxy is designed to provide a client with a single interface to multiple services. Service providers may offer the same services as other service providers as well as unique services. We have implemented a new grid service proxy that makes use of Wren measurements to help optimize the decision of which provider’s service to invoke.

The proxy service is an HTTP-level proxy service. The proxy parses the message to identify each unit of work as it is submitted and selects a particular endpoint that can provide the requested service. However, the proxy does not modify the request in any way and simply ensures that all messages corresponding to the particular unit of work are forwarded to the same endpoint. In the OGSA this service is most similar to a Message broker. Currently we do not implement any of the functionality of an OGSA Job Manager, but the services and algorithms we have implemented would work well within a Job Manager. However, because the application we utilize in these examples, INCOGEN’s VIBE toolkit, does not presently support OGSA functionality, we are able to achieve similar goals with an HTTP proxy. In this case, we configure VIBE with a list of the services that are available, and provide the address of our proxy for each service. VIBE then contacts the proxy to request the service, and the proxy transparently forwards the request to the best endpoint it is aware of to provide that service.

Figure 1 shows how the Wren packet trace facility and the Wren analysis module are integrated with a simple grid services proxy. Wren produces a series of available bandwidth measurements for each connection established on the

Table 1. Overhead of integrating Wren Kernel tracing with proxy.

	Pipeline 1 Avg time (s)	Pipeline 2 Avg time (s)	Pipeline 3 Avg time (s)	Pipeline 4 Avg time (s)
Proxy	5.2 ± 0.99	17.3 ± 1.85	10.5 ± 1.56	75.4 ± 0.83
Proxy + Wren Tracing	4.9 ± 1.09	18.5 ± 2.08	9.6 ± 0.89	75.6 ± 2.36

proxy machine. A connection is any TCP stream between the proxy and another host. These Wren measurements are integrated into a grid services proxy via a SOAP interface.

Available bandwidth measurements are gathered for each connection/path between the proxy and each server. The proxy stores available bandwidth information for each service registered with the proxy and uses this information to make decisions when a client requests a service that is offered by more than one server. In our implementation, the decision of which server to use is based solely on the available bandwidth on the paths between the proxy and the servers. To keep the available bandwidth measurements from becoming stale, the proxy will divert a small fraction of requests to alternate servers.

In the remainder of this section, we look at the overhead of integrating Wren with the grid services proxy, the accuracy of Wren measurements, and the measurement lag in providing the measurements to the proxy.

4.1. Impact of Wren Tracing on Proxy Performance

To determine the impact of Wren tracing on proxy performance, we evaluated the overhead incurred when running an example grid application. For this experiment, INCOGEN’s software VIBE (Visual Integrated Bioinformatics Environment) [4] will serve as the example grid application. VIBE is designed to provide workflow management and the point of interaction between users and their data. VIBE allows users to study a particular genome or explore genome sequence with sequence analysis tools.

In our experiments, we selected four example VIBE pipelines. In each pipeline, the client uploads a sequence to a VIBE server, asks the server to apply an algorithm to that sequence, and retrieves the results from the server. Table 1 compares the execution time of running the VIBE pipelines with Wren tracing and without Wren tracing. The values in the table represent averages with 95% confidence intervals. The results in the table show that there is no noticeable difference between execution time and that Wren tracing has a negligible effect on application performance.

4.2. Wren Measurements of Proxy Traffic

We demonstrate that Wren can provide real-time available bandwidth measurements when integrated in a grid

services proxy through experiments on a controlled-load/controlled-latency testbed environment. As validating experimental results on a WAN is difficult due to the inability to confirm the actual available bandwidth on the WAN during the experiment, we focus here on controlled-load testbed experiments.

We set up an experiment in which an application uses the proxy to communicate with a server across a LAN testbed. For this experiment, one iperf generated 50Mbps of cross traffic for the first 30 seconds and no cross traffic was present for the remaining 30 seconds. Therefore we know that there was 50 Mbps of available bandwidth from time 0–30 seconds and 100 Mbps of available bandwidth from time 30–60 seconds.

In the first 30 seconds of Figure 6, we see that our online Wren monitoring system produces measurements that accurately reflect the change in available bandwidth on the path. It is important to note that Wren can measure the available bandwidth even when the throughput of the application we are monitoring is not saturating the available bandwidth of the path.

4.3. Measurement Lag

The overheads associated with collecting and analyzing the Wren traces will impact the time it takes for the listening application to receive the Wren measurements. We define measurement lag as the delay from the time we start collecting and analyzing traces to the time those measurements are pushed to the listening application, in our case the grid services proxy, or any other observation thread.

The collector thread overhead is determined by the polling frequency, the time it takes to collect the traces from the kernel, and the time it takes to push traces to the analysis thread. The polling frequency specifies how often the collector pulls data from the kernel level buffers. By default this value is set to 1 second, but can be adjusted according to the traffic load and the application’s needs. The kernel level data is transferred to the user-level via the *copy_to_user* kernel function. Once the data is in a user-level data structure the information is pushed into the analysis thread’s trace queue.

The sources of overhead in the analysis thread are parsing traces and the application of the Wren algorithm. The incoming and outgoing packet information is first merged

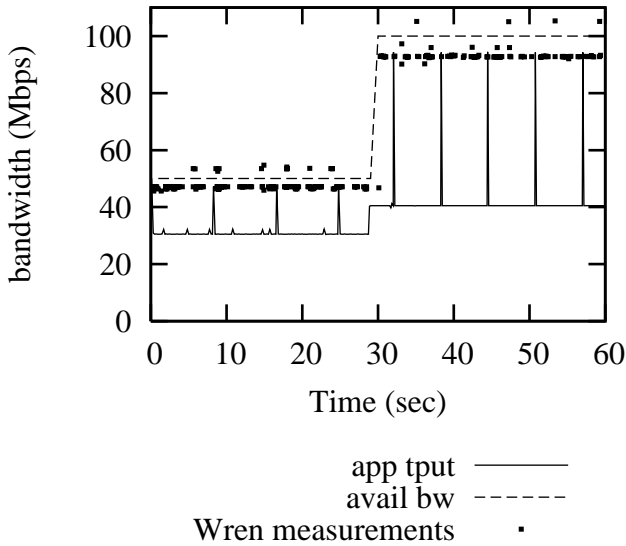


Figure 6. Wren measurements obtained from proxy traffic reflect changes in available bandwidth even when the monitored application’s throughput does not consume all of the available bandwidth.

into single packet views. These single packet views are then sorted by connection and passed into the Wren available bandwidth algorithm. The Wren algorithm is a one pass algorithm that calculates available bandwidth measurements that can be sent to listening applications.

To provide a sense of the total overhead associated with Wren, we set up an experiment to measure the lag when Wren pulls traces from the kernel every 1 second. We ran iperf on the machine Wren was monitoring to saturate the network link so that each call to Wren would return the maximum amount of packet information when polled. We determined the clock rate of the machine and counted the clock cycles to compute the time elapsed in the collector thread and the analysis thread. Table 2 shows the time spent in each thread. (Each number is an average of several runs with a 95% confidence interval.) These results show that the maximum lag time expected is approximately 0.25 seconds. This small measurement lag value means that the application will obtain the measurements in a timely manner.

Table 2 also quantifies the amount of processing time a node must spend to calculate Wren measurements. On a proxy that’s relaying a lot of data, but not doing a lot of computation, clearly this 0.25 second processing load is acceptable. Grid application nodes that do not saturate their link constantly can also tolerate the load of Wren monitoring, but for applications that are able to utilize both the full CPU

Table 2. Overhead of Collector and Analysis Threads.

Collector Thread processing time (excluding polling time)	Analysis Thread processing time
$.02852 \pm 0.00029$ s	0.21741 ± 0.00216 s

power and NIC bandwidth of a node, a choice must be made between allocating CPU time to the analysis thread when measurements are needed, transferring the data across the network to an analysis node, or simply reducing the amount of data used for the measurement.

To obtain a continuous view of the network bandwidth when an application is saturating the path, Wren should be used to read from the kernel buffer and analyze the traces at regular intervals. The user-level polling frequency determines how often the analysis thread will be run. Because Wren is designed to periodically capture smaller portions of continuous traffic and uses those to measure available bandwidth, the polling frequency does not affect the accuracy of the measurements. Complete traces are not necessary when an application is saturating the network path because additional fine grain measurements will consume more processing time than desired and provide little additional insight into the network conditions.

5. Proxy Server Selection

While available bandwidth is only one factor that must be taken into account when making the server selection decision, we ignore other factors, such as CPU power or cost for analysis in this paper. We motivate the proxy server selection by describing a scenario in which our example grid application, VIBE, would need to decide between replica service providers.

A cancer drug research group has sequenced genes that are expressed in cells from a specific type of skin cancer in humans (that is, genes that appear in cancerous cells) but not in healthy skin cells. To find out what these genes do, the research group compares the genes against locally stored results from the group’s previous experiments to see if there are similar genes expressed in other conditions. The group also has the option of running the same sequences against a national/international database such as *nr* at NCBI to see if other people have found the same gene expressed in other animals and/or in other conditions that they have not yet looked at. Both the local and remote databases will help the research group narrow down what the particular genes do and possibly help lead them to drugs or methods to disrupt the functioning of cancer cells.

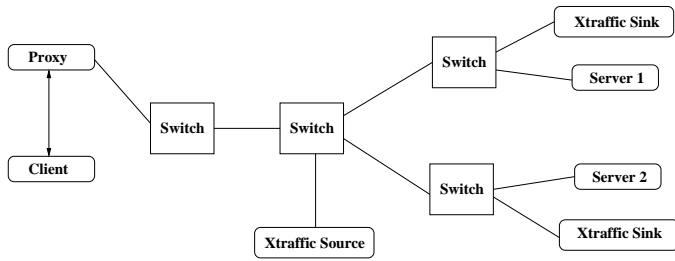


Figure 7. This testbed configuration allows us to adjust the amount of cross traffic on the paths between the proxy and the servers.

The decision the group must make is between using a small, locally located computation engine or a potentially more powerful remote resource, but with much more restricted upload bandwidth for their gene sequence. Obviously, the decision of whether to use the local or remote database should consider the network bandwidth available to both the local and remote databases. In the following experiment, we show how Wren measurements provide the necessary available bandwidth information to optimize the selection of server. In a real server selection process, this network performance information would be used in conjunction with other factors to select the optimal server.

5.1. Experiment

We created a testbed experiment in which we used a TCP traffic generator to simulate application traffic that would be produced by a grid application, such as VIBE. We used a testbed rather than creating a WAN experiment because confirming the actual available bandwidth in WAN environments is not always feasible and we have exact control over the network load in the testbed environment.

To demonstrate that Wren measurements can be used to help the proxy select the server that will optimize performance, we set up a system in which two servers provide the same services. Figure 7 shows the testbed configuration we used for this experiment. The client is only aware of the proxy, but the proxy is using Wren to monitor the available bandwidth to the two identical servers. Because Server1 and Server2 are located on separate paths, we can vary the amount of cross traffic, and thus the available bandwidth, between the servers.

In this experiment, we produced approximately 25 Mbps of cross traffic on the path between the proxy and Server1. Figure 8 shows that Wren correctly measures that there is approximately 75 Mbps of available bandwidth on the path. In this figure, we also see that even though the application throughput is not saturating the path, Wren can correctly

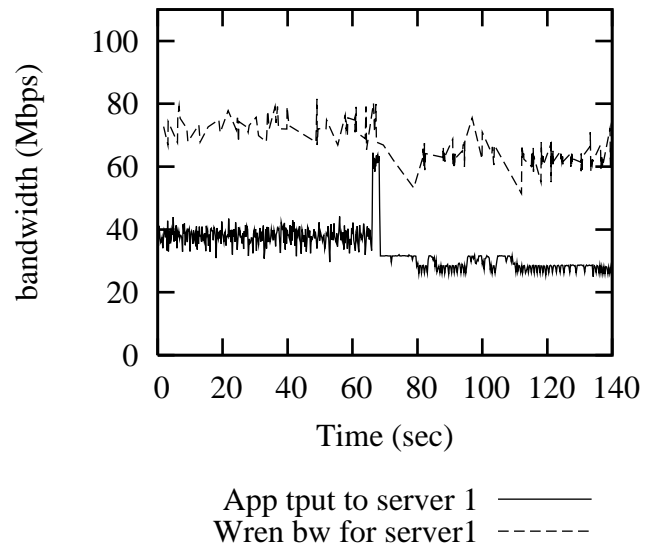


Figure 8. Wren measurements accurately reflect the available bandwidth of a path with 25Mbps of cross traffic even though the monitored application throughput is not saturating the path.

measure the available bandwidth.

On the path between the proxy and Server2, we produced approximately 75Mbps of cross traffic for the first 70 seconds of the experiment and 0 Mbps of cross traffic for the remainder of the experiment. In Figure 9, we see that the Wren can accurately detect the change in available bandwidth that occurs at time 70 seconds when 75Mbps of cross traffic is reduced to 0 Mbps of cross traffic. For this experiment, Server1 is the optimal server for the first 70 seconds because it has 75 Mbps of available bandwidth and Server2 only has 25 Mbps of available bandwidth. From time 70 seconds until the end of the experiment, Server2 is the optimal server because it has full available bandwidth while Server1 still only has 75 Mbps available.

In Figure 10, we show the servers selected by the proxy during the experiment are optimal according the available bandwidth between the servers and the proxy. This graph shows that the proxy can use Wren measurements to optimize network performance.

6. Conclusion

In this paper we have described how to transparently optimize grid server selection by incorporating Wren measurements into a grid services proxy. We have analyzed the efficiency of our trace collection components and demon-

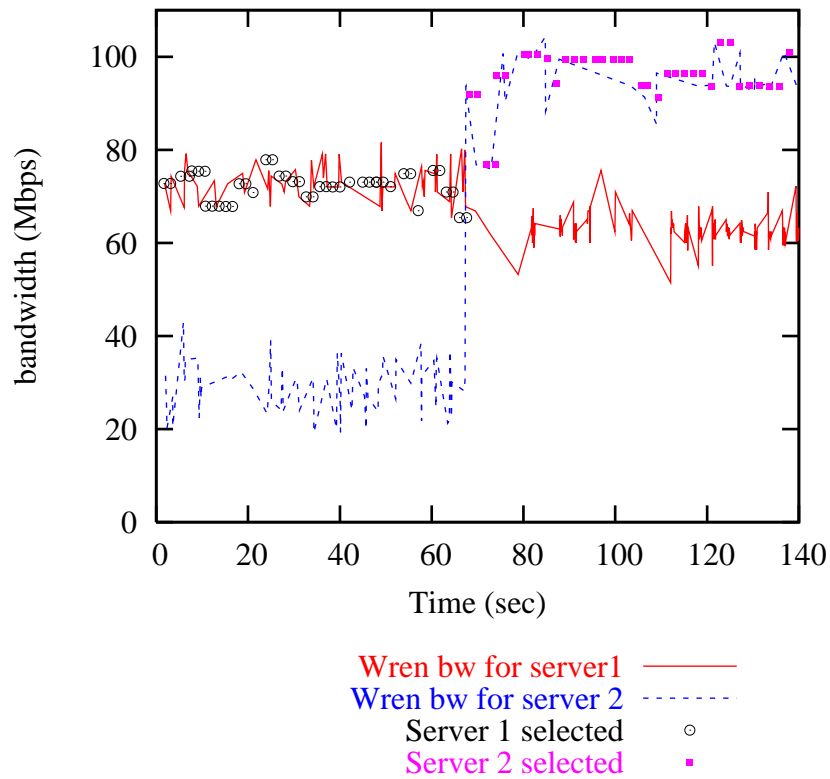


Figure 10. Wren measurements help the proxy select the server with the most available bandwidth to optimize overall performance.

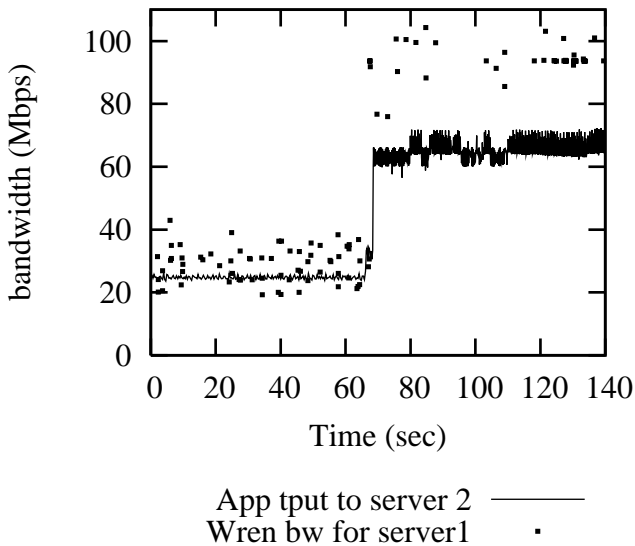


Figure 9. Wren measurements accurately detect the change in available bandwidth that occurs at 70 seconds when the 75Mbps of cross traffic is reduced to 0.

strated they are efficient enough to monitor traffic on gigabit networks and shown that they have minimal impact on the performance of the grid services proxy. We have shown that Wren measurements are accurate, even when monitored traffic is not saturating the network path, and Wren measurements can be delivered to interested applications with minimal lag. Finally, we experimentally demonstrated that a proxy can use Wren measurements to select the resource that will optimize network performance.

7. Acknowledgments

We would like to thank INCOGEN for allowing us to use their VIBE software toolkit. In particular, we would like to acknowledge the support provided by Jennifer Lowekamp and Dawn Cannan at INCOGEN. This research was supported in part by the National Science Foundation (award ACI-0203974) and the VSGC graduate fellowship. This work was performed, in part, using computational facilities at the College of William and Mary which were enabled by grants from Sun Microsystems, the National Science Foundation, and Virginia's Commonwealth Technology Research Fund.

References

- [1] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. V. Reich. The open grid services architecture, version 1.0. Technical Report GFD-I.030, Global Grid Forum, January 2005.
- [2] A. Gupta, M. Zangrilli, A. Sundararaj, A. I. Huang, P. Dinda, and B. B. Lowekamp. Free Network Measurement for Adaptive Virtualized Distributed Computing. In *20th International Parallel and Distributed Processing Symposium (IPDPS)*, April 2006.
- [3] N. Hu and P. Steenkiste. Evaluation and Characterization of Available Bandwidth Techniques. *IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling*, 2003.
- [4] INCOGEN. Vibe: Visual integrated bioinformatics environment. Whitepaper. www.incogen.com.
- [5] M. Jain and C. Dovrolis. Pathload: a Measurement Tool for End-to-end Available Bandwidth. In *Proceedings of the 3rd Passive and Active Measurements Workshop*, March 2002.
- [6] G. Jin and B. Tierney. Netest: A Tool to Measure the Maximum Burst Size, Available Bandwidth and achievable Throughput. In *International Conference on Information Technology Research and Education*, 2003.
- [7] B. B. Lowekamp, B. Tierney, L. Cottrell, R. Hughes-Jones, T. Kielmann, and M. Swany. Enabling Network Measurement Portability Through a Hierarchy of Characteristics. In *Proceedings of the 4th International Workshop on Grid Computing (GRID2003)*, 2003.
- [8] C. L. T. Man, G. Hasegawa, and M. Murata. A merged inline measurement method for capacity and available bandwidth. In *Passive and Active Measurement Workshop (PAM2005)*, pages 341–344, 2005.
- [9] S. McCanne and V. Jacobson. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *USENIX Winter*, 1993.
- [10] B. Melander, M. Bjorkman, and P. Gunningberg. A New End-to-End Probing and Analysis Method for Estimating Bandwidth Bottlenecks. In *Global Internet Symposium*, 2000.
- [11] R. Prasad, M. Murray, C. Dovrolis, and K. Claffy. Bandwidth Estimation: Metrics, Measurement Techniques, and Tools. In *IEEE Network*, June 2003.
- [12] V. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell. pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *Passive and Active Measurement Workshop (PAM)*, 2003.
- [13] S. Shakkottai, N. Brownlee, and kc claffy. A study of burstiness in tcp flows. In *Passive and Active Measurement Workshop (PAM2005)*, pages 13–26, 2005.
- [14] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, D. Snelling, and P. Vanderbilt. Open grid services infrastructure. Technical Report GFD-R.15, Global Grid Forum, 2004.
- [15] M. Zangrilli and B. B. Lowekamp. Using Passive Traces of Application Traffic in a Network Monitoring System. In *Proceedings of the Thirteenth IEEE International Symposium on High Performance Distributed Computing (HPDC 13)*. IEEE, June 2004.
- [16] M. Zangrilli and B. B. Lowekamp. Applying Principles of Active Available Bandwidth Algorithms to Passive TCP Traces. In *Passive and Active Measurement Workshop (PAM 2005)*, March 2005.