Dynamic Load Balancing of SSH Sessions Using User-Specific Selection Policies*

Paul Z. Kolano
NASA Advanced Supercomputing Division, NASA Ames Research Center
M/S 258-6, Moffett Field, CA 94035 U.S.A.

E-mail: paul.kolano@nasa.gov

Abstract

Ballast is a tool for balancing user load across SSH servers based on various criteria such as CPU load and system availability. It includes a load balancing client, a lightweight data server, and a data collection agent. Ballast is invoked as part of the SSH login process, so has access to the user name while making balancing decisions, which is not available in traditional load balancing approaches. This gives Ballast the unique ability to perform user-specific load balancing. This paper presents the Ballast architecture and examines the benefits of involving user-specific criteria in the balancing process. Two approaches for utilizing user information based on prediction and dynamic load metrics are analyzed using trace-based simulation and are found to have significant benefits when combined.

1. Introduction

Computational systems have limited resources that must be shared amongst the users operating on them. These limits exist for processors, memory, storage, I/O bandwidth, network bandwidth, among others, and are reached differently depending on the activity of users operating on the system. Since even operating near these limits affects the usability of a system, sites with high utilization typically spread users over a number of systems with similar capabilities such that each operates well within its absolute limits.

In such configurations, load is distributed across systems by some form of load balancer that selects a system for each request using a predefined balancing policy. Load balancing works well for protocols such as HTTP that generate uniform load across transient connections, but can break down when used with more persistent connections that have non-uniform load patterns over time and between requests. SSH is the prime example of this type of protocol. SSH sessions may last from seconds to months or more and at any time while active may consume from almost no resources up to the entire capacity of the system. Such widely varying behavior may easily result in imbalanced utilization over time where one system may become fully utilized while another sits idle.

This paper presents Ballast, an approach to **Bal**ancing **L**oad **A**cross **Sys**t**e**ms that was specifically designed for SSH. Unlike traditional general-purpose balancers that operate within the standard network stack, Ballast operates in conjunction with SSH client invocation so has access to the user name when making balancing decisions. This allows Ballast to support more advanced selection strategies that can be tailored to each individual user. In particular, historical user behavior can be utilized to predict the duration and resource consumption of each SSH session as well as to determine the load metrics that are most important to the given user. This information allows Ballast to provide significant improvements in balancing performance.

This paper is organized as follows. Section 2 discusses related work. Section 3 presents the Ballast architecture. Section 4 discusses the simulation methodology. Section 5 describes user-specific load balancing. Finally, section 6 presents conclusions and future work.

2. Related Work

The two most common approaches for load balancing are based on in-line network interception and DNS manipulation. In-line network load balancers such as the F5 Networks BIG-IP [4] utilize IP addresses as aliases and balance load by routing traffic destined for each IP alias to different destinations depending on load criteria and information within the intercepted packets. Layer 7 devices can inspect protocol traffic to determine protocol-specific details such as user names to make better balancing decisions.

*This work is supported by the NASA Advanced Supercomputing Division under Task Number ARC-013 (Contract NNA07CA29C) with Computer Sciences Corporation

Proc. of the 15th IEEE Symp. on Computers and Communications, Riccione, Italy, Jun. 22-25, 2010

This inspection is only feasible for unencrypted protocols, however, so user names cannot be extracted from encrypted protocols like SSH. DNS manipulation balancers such as the Cisco Systems Global Site Selector [1] dynamically adjust the IP addresses returned for host name aliases based on criteria such as availability and load. Unlike in-line balancers, DNS load balancers cannot perform application-specific load balancing since the only information they receive about any protocol is the host name.

Resource selection frameworks for grid computing such as Surfer [7] and Condor's Matchmaker [9] provide selection capabilities that are, in essence, a more generalized form of traditional load balancing. Whereas traditional load balancers select hosts based on the single criteria of load, these frameworks select resources based not just on load, but also on additional criteria such as CPU architecture and operating system type to maximize the suitability of resources to user applications. Smith [11] provides a service that may be integrated into these frameworks to minimize queue wait times by predicting the execution times of currently running tasks using historical application behavior.

Devarakonda et al. [2] show how the CPU time, file I/O, and memory usage of a process can be predicted based on historical behavior. When the pool of balanced resources is heterogeneous, however, traces of application behavior on one system may not be applicable to another due to varying performance characteristics. Iverson et al. [6] show how benchmark data collected on different system types can be used to adjust predictions of task execution times, allowing task observations to be shared across all systems.

Harchol-Balter et al. [5] utilize predictions of process lifetimes with preemptive process migration to significantly improve CPU load balancing over selections made at the time a process is created. Unfortunately, SSH processes are not easily migrated to different hosts without breaking the connection. Wu et al. [14], however, describe a modification that can be made to OpenSSH servers to migrate SSH sessions from a failed server to a recovery server. While this work was geared toward reliability, it may be suitable for session migration as well to take advantage of preemptive migration. Instead of migrating processes, Werstein et al. [13] suggest dynamically balancing child processes as they are created. While this approach is directly applicable to SSH session balancing, it has the potential to break applications that do not expect child processes to be executed on a different system than the parent.

An alternative to predicting load based on application performance or user behavior is predicting the future load of a system strictly from its load history. Dinda et al. [3] evaluate linear models for predicting future load from past load, but only predict 30 seconds into the future, which is not long enough for SSH session balancing. Yang et al. [15] provide an approach for predicting load further into the future based on tendencies of historical load data.

Load balancing strategies may be evaluated in different fashions. Zhou [16] simulates different strategies using CPU and I/O traces obtained from production systems. This is very similar to the approach discussed in Section 4. The other standard evaluation strategy is against artificially-generated probabilistic loads such those demonstrated by Livny et al. [8]. Trace-driven loads offer simulation under more realistic conditions, but may not cover all scenarios appropriately, while artificial loads may test more scenarios, but many of those scenarios may not occur in practice.

3. Architecture

Ballast is a framework that was specifically designed for SSH load balancing by taking advantage of the aliasing and proxying capabilities available in most SSH clients. Within a traditional SSH bastion architecture [10], where internal resources can only be accessed from the outside via a hardened bastion system, Ballast is completely transparent to users and does not require any external modifications. Bastions and internal hosts must have the Ballast client and a pipe-based TCP relay utility such as netcat installed, which can be easily managed across internal hosts by configuration management tools.

Figure 1 shows the architecture of Ballast, which consists of a Ballast agent on each balanced host, a Ballast server on one or more data servers, and a Ballast client on each bastion and internal host from which the user might access balanced hosts. The Ballast agent periodically collects system load information and sends it to the Ballast server. The server aggregates the load data received from all agents and stores it in a suitable form for making balancing decisions. When the user invokes SSH to a host alias designated for balancing, SSH triggers the Ballast client, which contacts the server to resolve balancing aliases to actual balanced host names. The server consults its data and returns one or more hosts, one of which the client connects to via netcat (or equivalent), after which the login proceeds normally. The following sections discuss these components in more detail.

3.1. Ballast Agent

The Ballast agent runs on each balanced system and periodically collects various system and process measurements from the linux process file system. System measurements include CPU load, physical and virtual memory usage, I/O statistics, network statistics, and uptime. Process measurements include, among others, user name, process name, process ID, parent process ID, start time, accumulated CPU time, resident set size, and virtual memory size. Load data

Proc. of the 15th IEEE Symp. on Computers and Communications, Riccione, Italy, Jun. 22-25, 2010

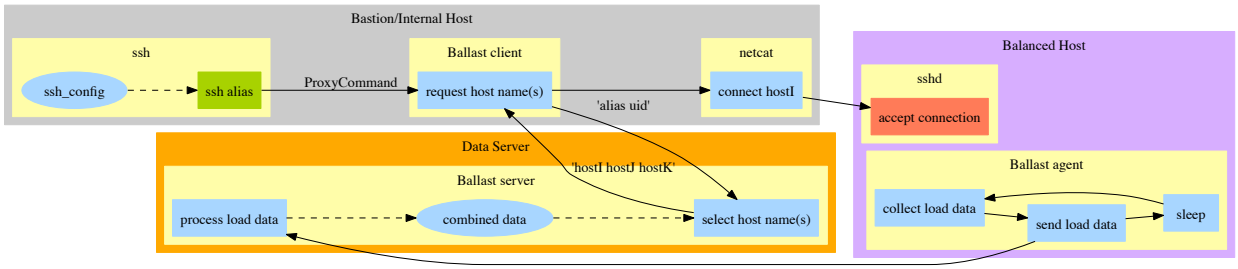

Figure 1. Ballast architecture

is sent back to the Ballast server via a TCP connection, where it is used to make balancing decisions.

A periodic push of data to the server was used in favor of an on-demand pull from all agents at the time of balancing for a number of reasons. A push-based model yields better response time as the server already has access to the data when making balancing decisions so does not have to poll all agents while possibly waiting for timeouts. This benefit becomes more significant as the number of balanced hosts increases. Pushing also scales better as the rate of balancing requests increases since the data update rate is fixed.

Pushing has some disadvantages compared to an on-demand pull. First, load data will be slightly stale at the time balancing decisions are made. Ballast uses a default update rate of once a minute, meaning decisions will be made on data that is an average of 30 seconds old. It has been shown [3, 15], however, that system load within this timeframe is predictable with reasonable accuracy, hence stale data need not actually be stale. The second disadvantage is that load data must be continuously collected even when the balancer is not being used. The agent consumes only 0.04 CPU seconds per invocation, however, hence adds only fractions of a percent to CPU load. The continuous collection also allows user profiles to be accurately updated for use in user-specific balancing as will be described in Section 5.

3.2. Ballast Server

The Ballast server is responsible for accepting balancing requests from clients and returning the most suitable hosts for the given alias and user based on a chosen load balancing strategy applied to the data collected by agents. Balancing requests are of the form "alias uid", which together with client host IP address from the connection details, allow the server to balance on any combination of balancing alias, invoking user, and invoking host.

The server utilizes data received from agents to update current system loads and to track cumulative averages of session and process duration, CPU load, memory usage, etc. per user as well as the current processes active on each system for use within user-specific strategies, discussed in Section 5. To facilitate high availability, each agent sends data only to the first available server, with servers synchronizing consolidated agent data periodically with peers. This supports both IP takeover as well as connection retry models.

The server is implemented in Perl with balancing strategies being functions that are given the alias, uid, and invoking host as parameters. The basic strategy template is to iterate over all available hosts for a particular alias and compute the user-specific load metric for each based on the load data stored in the server's hash tables. The hosts with the lowest values are then returned back to the client.

3.3. Ballast Client

Ballast is an SSH-specific solution that relies on the aliasing and proxying features available in most SSH clients. To simplify the discussion, the dominant client implementation, OpenSSH, will be assumed in the remainder. Aliasing (using the ssh_config "Host" directive) allows host names to be specified on the SSH command line that need not resolve to IP addresses. Proxying (using the ssh_config "ProxyCommand" directive) allows a specific command to be executed to form the connection with the target host.

In Ballast, aliasing is used to specify load balancing aliases together with proxying that redirects connections originally destined for balancing aliases to the Ballast client. Before connecting to an actual host, the client connects to the Ballast server to obtain the most suitable set of actual hosts for the given alias and then connects to one on the SSH port using netcat (or equivalent). The necessary SSH client configuration can be specified in a global location by the administrator, allowing Ballast to be completely transparent to users.

Because the Ballast client is directly (though transparently) invoked by users, the user for which the balancing decision is being made is known at the time of invocation. This information is forwarded to the Ballast server, which allows it to make user-specific decisions that are not possible with other load balancing approaches. The benefit achievable with this additional information is the topic for the remainder of the paper.

4. Simulation Methodology

To evaluate different balancing strategies across a varying number of hosts, trace-driven simulation [16] was used with load data collected from eight computational front-ends of the Pleiades system at NASA Ames Research Center, which was recently ranked as the sixth fastest computer system in the world [12]. Ballast is currently utilized for distributing users across the Pleiades front-ends. 897,823 samples were taken at one minute intervals over a three month period with each sample corresponding to the load data normally collected by Ballast agents.

4.1. Simulation

After the data was collected, SSH sessions and related child processes were extracted from the last two months of the process data. At each sample time, a record was created that indicated the CPU time and total memory usage of each process within the SSH session at that moment. The resulting session history contained 19,212,588 samples of 27,300 SSH sessions over 439 users observed during the collection period. This session history was then used as the basis for a simulator that produces system load measurements at each time in the collection period for a given load balancing strategy and number of hosts. The simulator works by traversing the session history chronologically. Whenever a new SSH session is observed in the history, the simulator chooses a host based on the balancing strategy. Load measurements are computed at each point based on the difference in the processes and the CPU/memory metrics between samples.

Figure 2 shows the actual CPU time per minute observed during the original collection period. Figure 3 show the simulated CPU time per minute using the original system selection and original number of hosts. As can be seen, the curves for the actual load and simulated load have similar shapes indicating that the simulator generates reasonable approximations of actual behavior. Some variation is expected due to activity on the systems that could not be directly associated with any SSH session, hence not simulated, or activity that could not be properly measured using an interval of one minute.

4.2. User Profiling

To facilitate the specification of user-specific balancing strategies, profiles of typical user behavior were constructed based on the first month of the three month sample period. The sample data during this period contained 8,649,828 samples of 10,041 SSH sessions over 310 users.

Table 1 shows a statistical summary of the number of sessions and averages of CPU time, total memory, and duration observed across all users over the profiling period.

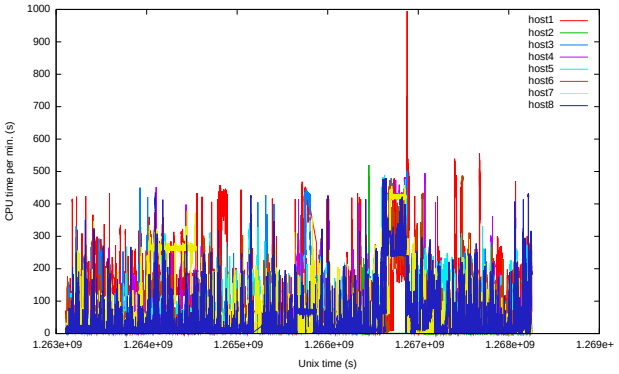

Figure 2. Original system load

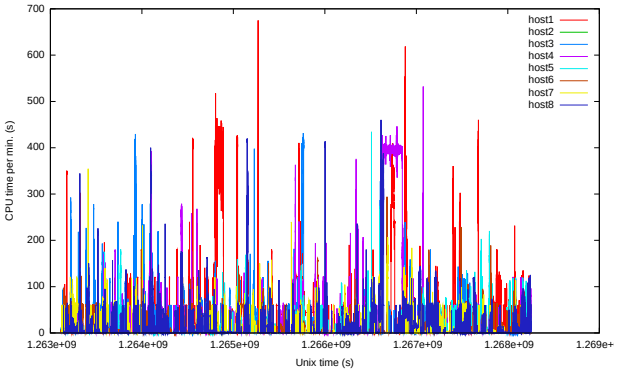

Figure 3. Simulated system load

Of particular note are the mean and max columns, which represent the resources consumed by the typical and heaviest users in each category. As can be seen, there is a wide variation between the two. In particular, the average CPU time and average total memory differ by almost two orders of magnitude. In Section 5, it will be shown how the differentiation of users of different types can be used to provide better balancing for each.

5. User-Specific Load Balancing

Unlike traditional load balancers, Ballast has the ability to adjust its balancing strategy based on the invoking user. A primary motivation of this research was to determine if a balancer that has access to this additional information can distribute load better than one that does not.

5.1. Predictive Strategies

As part of this assessment, an additional goal was to determine if strategies employing predictive balancing based on historical user behavior had measurable benefit. To measure effectiveness, four standard load balancing strategies,

Proc. of the 15th IEEE Symp. on Computers and Communications, Riccione, Italy, Jun. 22-25, 2010

Metric \\ Stat	Mean	Median	Std. Dev.	Min.	Max.
Avg. CPU time	8.97 m	9.12 s	44.1 m	0.00 s	8.12 h
Avg. tot. mem. (GB)	1.26	0.18	4.65	0.01	66.6
Avg. duration	52.5 h	6.52 h	5.07 d	1.00 s	35.4 d
Sessions	32	11	132	1	2036

Table 1. User statistics during profiling period

two predictive strategies, and two user-specific predictive strategies were run through the simulator with a varying number of hosts against load measures of CPU time, total memory usage, and a composite metric of the product of the two. The standard strategies evaluated included:

- *Random* - Choose the system at random.
- *Round robin* - Choose systems sequentially.
- *Least users* - Choose the system with the lowest number of users.
- *Least load* - Choose the system with lowest load.

The predictive strategies were:

- *Predictive* - Choose the system with the lowest sum of the average load per sample interval of the users currently on each system (the *predicted load*).
- *Predictive least load* - Choose the system with the lowest sum of the actual and predicted loads.

Finally, the user-specific predictive strategies were:

- *Predictive average overlap* - Choose the system with the lowest sum of the actual and predicted load with each user's predicted contribution reduced in proportion to the ratio of the invoking user's average session duration to that of their own (the *predicted average overlap load*). No reduction is made for users with an average session duration greater than the invoking user's. The idea of this strategy is that the invoking user will be affected less by the future activity of another user if they typically stay logged on for longer.
- *Predictive overlap* - Choose the system with the lowest sum of the actual and predicted average overlap load, but with the predicted contribution reduced even further by using the average session duration minus the already elapsed session duration for each user (the *predicted overlap load*). The predicted contribution of users who have been on the system longer than their average session duration will be zero. The idea of this strategy is that other users may affect the future activity of the invoking user even less if they are reaching the point at which they normally log off.

The measure chosen to evaluate the load balancing strategies is the average of the standard deviations of the load across all systems at each point in the simulation period. Since the total load across all systems at each point is fixed by their original sampled values, the standard deviation at each point indicates how evenly the load has been distributed. A value of zero indicates a perfect distribution with balance decreasing as the standard deviation increases.

Figures 4, 5, and 6 show the averages of the standard deviations of each one minute sample interval when balancing CPU time, memory usage, and composite CPU/memory load, respectively, across 2-12 systems. The purely predictive strategy, which was included only for comparison purposes, performed the worst on average of any of the strategies in all three cases. The other predictive strategies were only able to outperform the best-performing traditional strategy (least load) in 13/33 CPU load cases, 15/33 memory usage cases, and 4/33 composite load cases. The average performance loss for the CPU and memory cases was minimal, however, at 0.2% and 0.3%, respectively. The average loss for the composite case was more significant at 8.6%. Among the predictive strategies, predictive overlap performed slightly better on average than predictive least load, which performed slightly better than predictive average overlap, with each beating least load in 14, 11, and 7 of the 33 cases, respectively. While the benefits of prediction were inconclusive in this portion of the study, the next section will demonstrate significant improvements achieved with predictive strategies when paired with dynamically selected load metrics.

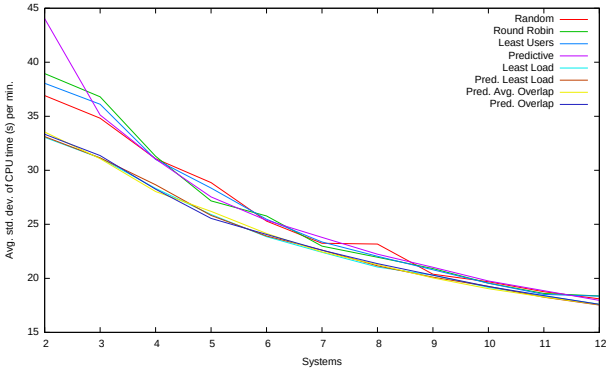

Figure 4. Avg. std. dev. of CPU load

5.2. Dynamic Load Metrics

While the predictive overlap strategies of the previous section demonstrated how the invoking user's profile could be incorporated into different balancing strategies, the strategies themselves were based upon on a single predetermined metric of either CPU, memory, or composite load.

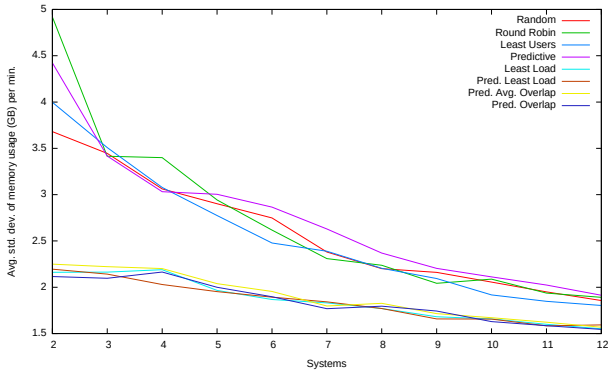

Figure 5. Avg. std. dev. of memory usage

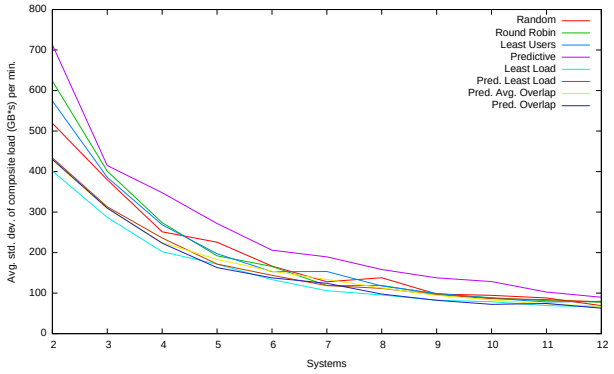

Figure 6. Avg. std. dev. of composite load

As illustrated in Section 4.2, users have widely varying usage patterns with correspondingly varying requirements, hence the metric most important to one user is not necessarily the one most important to another. Hence, a balancer that selects systems using the same metric for all users may direct users to systems that are inferior for their purposes.

With the ability to balance based on the invoking user comes the ability to balance based on the metric most important to each user. This section examines the potential benefits of selecting the load metric on a per user basis. In particular, we divide users into four classes depending on whether they have historically high CPU utilization, high memory utilization, both high CPU and high memory utilization, or none of the above. The 75th percentile was used as the dividing line between classes, thus users at or above the 75th percentile of one or both of CPU and memory in their profiles were classified accordingly. The best four strategies of the previous section were then simulated with the load metric being chosen dynamically based on the class of the invoking user and compared against standard least load. CPU time was used as the default metric.

Figures 7, 8, and 9 show the average CPU time, memory usage, and composite load, respectively, experienced by users in the corresponding classes over the course of their sessions. Note that both high CPU utilization users and normal users are represented in Figure 7 as CPU time was the default metric. While least load with CPU and composite metrics produced slightly lower average CPU and composite loads, respectively, than the dynamic versions of predictive least load and predictive average overlap, the other 32/36 cases resulted in higher average load. By using the same metric for all users, hosts with low values are given to users in other classes who don't always need low values of that metric, thereby clogging up systems for those that do.

Dynamic metrics achieved significant decreases in the average loads that each user cares about most. Adding dynamic metrics to least load provided average CPU, memory, and composite load decreases of 14%, 29%, and 21%, respectively, over standard least load. The best performing strategy was dynamic predictive overlap, which provided average CPU, memory, and composite load decreases of 16%, 33%, and 19% over least load and 2.6%, 1.0%, and 2.3% over the other dynamic strategies. From a traditional global perspective of standard deviation across systems, dynamic predictive overlap was actually 0.3%, 40%, and 4.4% less balanced than least load in the CPU, memory, and composite cases, respectively. As can be seen, however, the global view is deceptive as the distribution was significantly better for individual needs.

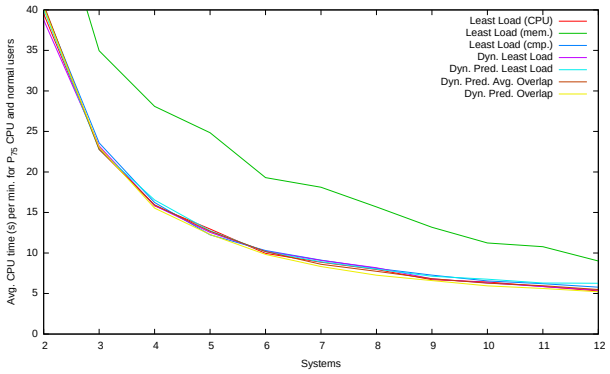

Figure 7. CPU load seen by P_{75} CPU users

Note that dynamic load metrics can be partially emulated with traditional load balancers by using metric-specific host aliases (e.g. host-cpu or host-mem). This quickly becomes impractical, however, as the number of aliases is exponential (2^n) in the number of metrics. With just CPU and memory, there must be 4 aliases. Other useful metrics are network and I/O load, bringing the total to 16. Other characteristics such as differing user accessibility, software license availability, different file system mounts, etc. would put the number of aliases out of reach for any user to remember. Add to this that users may not even know what metrics matter most for their applications nor how these requirements

Proc. of the 15th IEEE Symp. on Computers and Communications, Riccione, Italy, Jun. 22-25, 2010

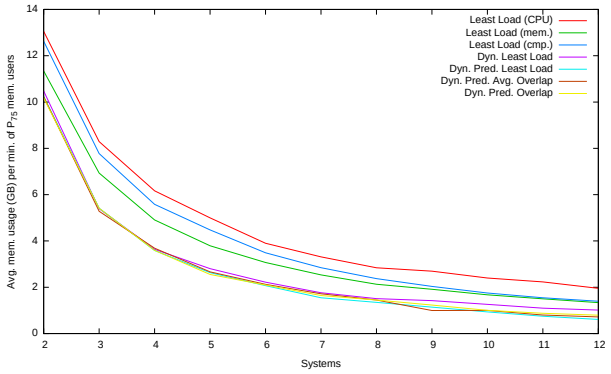

Figure 8. Mem. load seen by P_{75} mem. users

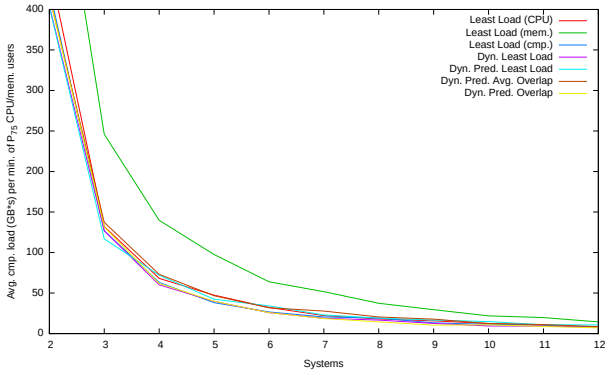

Figure 9. Cmp. load seen by P_{75} C/M users

may change over time. With Ballast's user-specific load-balancing, profiles are automatically derived and dynamically adjusted over time to match current user activity.

6. Conclusions and Future Work

This paper has described Ballast, an approach to **Bal**ancing **L**oad **A**cross **Sy**s**t**ems. The main contribution of Ballast over other balancing approaches is its ability to perform user-specific balancing of SSH sessions. By tailoring system selections specifically to the invoking user, the utility to each user can be maximized instead of choosing a system that may be best according to a common metric, but not ideal for the user's needs. Through the use of prediction and dynamic load metrics, Ballast achieved average load decreases of 16%, 33%, and 19% for CPU, memory, and composite loads, respectively, over a least load policy in the metrics most important to each user. The end result is that access to the invoking user during system selection has significant benefits for SSH load balancing, which cannot be achieved with other approaches.

There are several directions for future research. Currently, data collection relies on periodic samples of the linux process file system. Precision can be increased by using kernel auditing facilities, although this may induce additional overhead. Profiling per user application load patterns would also improve prediction accuracy by taking both user and application into account. For example, the duration of a session that immediately runs scp is likely to be less than that of a normal interactive session.

The per user I/O stats in linux 2.6.20+ kernels are not available on the systems Ballast was developed for, but should be integrated into the Ballast agent and server processing. Per process network load would also be useful, but is not yet available in linux kernels. It may be possible to estimate this load by analyzing process activity during various patterns of system network load, but further investigation is required. Finally, user-specific balancing techniques should be investigated for protocols besides SSH.

7. Acknowledgments

Special thanks to Ernst Kimler who designed and implemented the initial version of the Ballast agent and to Herbert Yeung who helped deploy Ballast onto production systems.

References

[1] Cisco Systems Global Site Selector. `http://www.cisco.com/web/go/gss`.
[2] M.V. Devarakonda, R.K. Iyer: Predictability of Process Resource Usage: A Measurement-Based Study on UNIX. IEEE Trans. on Software Engineering, vol. 15, no. 12, Dec. 1989.
[3] P.A. Dinda, D.R. O'Hallaron: An Evaluation of Linear Models for Host Load Prediction. 8th IEEE Intl. Symp. on High Performance Distributed Computing, Aug. 1999.
[4] F5 Networks BIG-IP. `http://www.f5.com/products/big-ip`.
[5] M. Harchol-Balter, A.B. Downey: Exploiting Process Lifetime Distributions for Dynamic Load Balancing. ACM Trans. on Computer Systems, vol. 15, no. 3, Aug. 1997
[6] M.A. Iverson, F. Ozguner, L.C. Potter: Statistical Prediction of Task Execution Times Through Analytic Benchmarking for Scheduling in a Heterogeneous Environment. 8th Heterogeneous Computing Workshop, Apr. 1999.
[7] P.Z. Kolano: Surfer: An Extensible Pull-Based Framework for Resource Selection and Ranking. 4th IEEE/ACM Intl. Symp. on Cluster Computing and the Grid, Apr. 2004.
[8] M. Livny, M. Melman: Load Balancing in Homogeneous Broadcast Distributed Systems. Computer Network Performance Symp., Apr. 1982.
[9] R. Raman, M. Livny, M. Solomon: Matchmaking: Distributed Resource Management for High Throughput Computing. 7th IEEE Intl. Symp. on High Performance Distributed Computing, Jul. 1998.
[10] M.J. Ranum: Thinking About Firewalls. 2nd World Conference on Systems Management and Security, Apr. 1993.
[11] W. Smith: Prediction Services for Distributed Computing. 21st IEEE Intl. Parallel and Distributed Processing Symp., Mar. 2007.
[12] TOP500 Supercomputing Sites, Nov. 2009. `http://www.top500.org/lists/2009/11`.
[13] P. Werstein, H. Situ, Z. Huang: Load Balancing in a Cluster Computer. 7th IEEE Intl. Conf. on Parallel and Distributed Computing Applications and Technologies. Dec. 2006.
[14] H. Wu, A. Burt, R. Thurimella: Making Secure TCP Connections Resistant to Server Failures. 19th Annual Computer Security Applications Conf., Dec. 2003.
[15] L. Yang, J.M. Schopf, I. Foster: Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments. Supercomputing 2003, Nov. 2003.
[16] S. Zhou: A Trace-Driven Simulation Study of Dynamic Load Balancing. IEEE Trans. on Software Engineering, vol. 14, no. 9, Sept. 1988.