Achieving Performance Consistency in Heterogeneous Clusters

Changxun Wu and Randal Burns
Department of Computer Science
Johns Hopkins University
{wu,randal}@cs.jhu.edu

Abstract

Hash-based randomization is a powerful technique used in clusters and distributed systems for load management. It offers uniform distribution, efficient addressing, little shared state, and scalability. However, simple hash-based randomization is unable to deal with skew and heterogeneity and, therefore, cannot achieve load balance in many environments. Virtual processors have been proposed as a solution to simple randomization's problem. We evaluate an alternative load management scheme for heterogeneous, shared-disk clusters. Our scheme directly tunes hash-based randomized load placement using a technique called adaptive, non-uniform (ANU) randomization [40] and compares favorably to the virtual processor approach. It provides the load balancing benefits of virtual processors with less shared state. It also automatically adapts to workload and cluster configuration changes, such as failure and recovery and adding or removing servers, without human involvement. Experimental results show that our scheme outperforms virtual processors and performs comparably to prescient load-balancing algorithms. They also show that our system maintains consistent performance across all servers while moving a minimal amount of load.

1 Introduction

Simple hash-based randomized load management schemes balance load effectively in homogeneous environments and incur very small overhead, which makes them appealing to traditional clusters and distributed systems. These schemes also provide an efficient addressing scheme. Because hashing is deterministic and requires no I/O operations, lookup operations can be quickly completed by one or a few hash computations, avoiding the need of a replicated lookup table that would increase the size of shared state among system nodes. Therefore, these schemes provide good scalability and fit well into the decentralized architecture of clusters and distributed systems.

The trend of building clusters on commodity hardware creates new challenges for cluster management and makes simple randomized load management schemes no longer suitable. Clusters are built increasingly with heterogeneous commodity components, and large-scale grids are often built upon multiple existing clusters that have different configurations. Such diversity and scale imply that high performance computing systems are becoming more dynamic, and, modern cluster load management systems should efficiently support heterogeneous hardware environments and changing system configurations. Developed with the assumptions that nodes are homogeneous and workload units are uniformly distributed, simple randomized schemes are not able to deal with hashing skew and heterogeneity in current cluster and distributed systems.

Virtual processors [17, 28] have been adopted as a solution to handle randomization variance and heterogeneity in many systems [9, 17, 28, 31, 35]. A virtual processor is an abstraction of some processing capacity and appears as a real server. In reality, many of these virtual units are mapped to a single processor. Systems dynamically map virtual processors to physical servers to balance load. Such mapping copes with load skew resulting from variance and heterogeneity. However, virtual processor schemes do not provide efficient addressing. They maintain the address information of each individual virtual processor, which can produce large shared state.

We have developed a technique called adaptive, non-uniform (ANU) randomization [40] for the issue of load management in heterogeneous environments. ANU randomization is derived from the SIEVE adaptive hashing strategy of Brinkmann *et al* [5], and, is suitable for any cluster system that partitions workload [2, 15] and has relatively short tasks, such as Web serving and file metadata serving. It makes randomized load placement tunable by adding a layer of abstraction between servers and workloads. Workloads are hashed to a unit interval. Servers are assigned to non-overlapping regions of the same unit interval and serve the workload partitions that

fall into their assigned regions.

ANU randomization compares favorably to virtual processors. It provides all the load balancing benefits with less shared state. Instead of keeping the address information of each individual virtual processor, our technique keeps only the mapped region information of each server and, therefore, reduces the size of shared state. Our technique also has several other virtues. It ensures performance consistency for application workload over any server in a heterogeneous cluster. It preserves load locality when adapting to cluster configuration changes. ANU randomization makes no assumptions about application behaviour or server capability. Servers are dynamically interchangeable and reconfigurable without negatively affecting performance of applications, facilitating the trend of building "clusters on demand" [7]. For example, the same server might be deployed in different clusters at different times during the same day or hours.

Simulation results show that ANU randomization performs comparably to a prescient system and provides consistent performance for application workload on any server in heterogeneous clusters with minimal load movement. Simulation results also demonstrate that virtual processor systems need to maintain a much larger shared state to achieve similar performance to ANU randomization.

2 Related Work

Much work has been done in the area of load management in clusters and distributed systems. A number of dynamic load management techniques are designed for parallel systems and homogeneous clusters [10, 37, 39, 41, 42]. Workload is transferred from heavily loaded servers to lightly loaded ones. Another family of techniques [36, 43, 44] take into account server heterogeneity but require all servers to periodically broadcast load and available capacity. These techniques assume knowledge of the capacity of any given server. For example, Zhu *et al* [44] use knowledge of server capacity and employ a metric that combines available CPU cycles and disk capacity to select a node for processing an incoming request.

Randomization is a powerful technique for load management in clusters and distributed systems [24]. Many systems, especially peer-to-peer systems [32, 33], rely on randomization, or more specifically pseudo-random hashing, to uniformly distribute workload. There are challenges in balancing load dynamically in global scale networks. Message exchanges may have to travel across the Internet, which makes it difficult to move load. Also, herds of tasks from many nodes may simultaneously move together to a node that previously had available capacity [25]. Therefore, these systems do not actively balance load. They use simple randomized load placement, which balances load in practice while avoiding message exchanges between the system nodes [24, 26]. Although simple randomization works well in certain environments, our experiments indicate that it cannot support extreme workload and server heterogeneity.

Virtual processors are another technique widely used for load balancing in parallel systems and conventional clusters [17, 28]. Each virtual processor is assigned a small portion of the entire workload and the system dynamically maps these virtual processors to physical processors to achieve load balance. Peer-to-peer systems have adopted the virtual processor approach to manage load on peer nodes as well [9, 31, 35]. However, virtual processor-based load balancing requires a larger shared state to achieve similar performance to our system.

Similar to our approach, the distributed lookup schemes used in some peer-to-peer systems [33, 35] also map values (keys and nodes in these cases) to an artificial one-dimensional space, and try to match them by their offsets within this space. However, these schemes assume that peer nodes are homogeneous and objects have the same size [31]. By themselves, these schemes cannot handle server or workload heterogeneity.

Many systems redirect requests within a cluster to dynamically balance load. This is a popular form of load balancing suitable for systems in which any server can handle any request. Examples include DNS rotation [6, 18] and request routing in Web servers [3]. Similarly, I/O systems use striping to distribute a large I/O request across many disks in a load-balanced fashion [13]. This approach applies when jobs are divisible and large, and, thus, are not suitable for short tasks. Striping is frequently used in multimedia servers [34]. There also has been research in the area of functional decomposition [1, 2]. Instead of dividing workload among cluster servers, the system places different functions at different servers. For example, an interactive file metadata workload can be split from a throughput oriented large-file I/O workload [2].

There is a long history of load balancing through process migration [4, 14, 21, 30] in which active processes are transferred among computers. Process migration reorganizes the assignment of long running jobs among a cluster of computers. These techniques do not apply to Web serving and file serving, which consist of relatively short tasks.

3 Clustering and Performance Issues

Heterogeneity and variability in clusters makes them vulnerable to performance inconsistency and partially accounts for high administration costs, which often dominates ownership costs [12, 19, 38].

Several factors contribute to the performance vulnerabilities of clusters. Clusters must adapt to changing workloads and hot spots. Conventional cluster load management

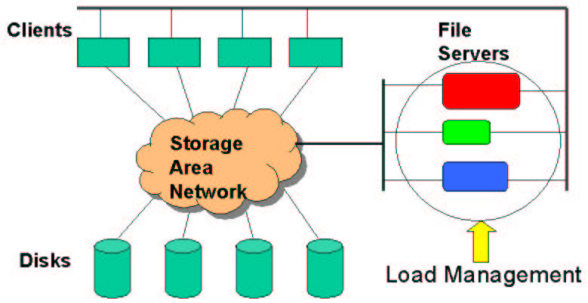


Figure 1. Shared-disk cluster system architecture.

systems focus on workload heterogeneity and are not sensitive to server heterogeneity. As computing systems shift from parallel homogeneous hardware to heterogeneous commodity configurations, cluster nodes could easily deliver inconsistent performance for application operations. Such inconsistency could negatively affect the overall performance of applications and makes clusters vulnerable to load misplacement. The trend of integrating clusters into computational grids, together with the relatively high failure rate of commodity hardware, makes the situation even worse, because it implies frequent configuration changes.

Human administration costs of cluster systems built from commodity components are a large and growing issue. For example, Feng *et al* reported that administration cost dominated the total cost of ownership for a 24-node cluster over a four-year period [11]. In most traditional cluster management systems, human intervention is required in the cases of system configuration changes such as adding/removing servers and failure and recovery. To reduce high administrative cost, it is crucial to have an adaptive, self-tuning cluster management system that integrates automated management functions. Conventional cluster management systems need to be upgraded with automated, efficient heterogeneity and dynamic configuration support to ensure good performance and low cost.

To address the problems discussed above, the objective of this work is to provide an efficient load management system for heterogeneous, shared-disk clusters. The system aims to maintain good and consistent performance for applications in heterogeneous, shared-disk clusters without sacrificing overall throughput. It also operates without prior knowledge of heterogeneity, automatically adapts to configuration changes, and minimizes load movement during rebalancing due to cost.

A brief review of the architecture of shared-disk file system clusters [16, 22, 23, 29] and their workload characteristics helps to facilitate our discussion and motivate our solution. A shared-disk file system cluster usually uses a single global namespace, which is partitioned into file sets. A file set is a subtree of the global namespace and also the indivisible unit of workload assignment and movement. Many shared-disk file system clusters distinguish between data and metadata, which are stored and accessed separately. The shared disks hold file sets and metadata of the file sets are assigned to file servers. In a typical access, client sends a metadata request to a file server. The server sends the location information and file handler of the specified file(s) back to the client. Then the client fetches data directly from the disk across the storage area network (SAN). This architecture separates metadata workload from data workload and targets them to file servers and shared disks respectively. File servers are loaded with single class of metadata operations and do not serve sequential or large file I/O, which goes to shared disks. Figure 1 shows the architecture of a typical shared-disk file system cluster.

Our system focuses on managing load on file servers. In shared-disk file systems, file servers are recoverable resource. Imbalance in file servers adversely affects overall system performance, because clients acquire metadata prior to data. Clients blocked on metadata may leave the high bandwidth SAN underutilized. Our system does not address load management issues in shared disks, which is a separate problem by itself and requires different strategies. Although our load management scheme was initially designed for shared-disk file system clusters, it is suitable for any cluster system that partitions workload and has relatively short tasks.

4 ANU Randomization

We use a technique called adaptive, non-uniform (ANU) randomization [40] to manage load placement. ANU randomization is based on the SIEVE adaptive hashing technique described by Brinkmann *et al* [5]. The flexibility of ANU randomization comes from an extra level of abstraction in the workload to server mapping, where it employs a unit interval to which it maps both workload and servers. The ability to dynamically update the mapping of the servers allows us to tune the system in a fashion that is not possible when mapping workload directly to servers (simple randomization).

ANU randomization first maps file sets to offsets in a unit interval (Figure 2) by hashing the unique name of each file set. We call such offsets in the unit interval *hashed offsets* of file sets. The unique name of a file set is specific to clusters, such as a pathname or content fingerprint. The unit interval is partitioned into multiple sub-regions of the same size. We assign to each server one or more sub-regions. A server completely occupies all but one assigned sub-region, which may be partially occupied. In the following discussion, we call these sub-

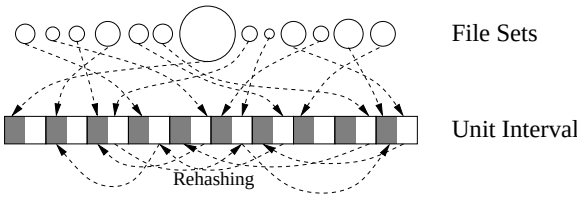


Figure 2. Servers are assigned sub-regions of the unit interval. File sets of different size, representing different workloads, are hashed into this interval. File sets not mapped to servers by the first hash are re-hashed until assigned.

regions *partitions* and we call the segments within the unit interval occupied by a server its *mapped region*. Each server is then assigned file sets whose hashed offsets lie within the server's mapped regions. ANU randomization balances load by changing the sizes of the server mapped regions based on some simple performance metric. By repartitioning the interval and scaling the server mapped regions, this technique copes gracefully with hardware changes, such as adding and removing servers.

For a system with k servers, we divide the unit interval into $2^{\lceil \lg k \rceil + 1}$ partitions of equal size. The system does not assign servers to all portions of the unit interval. Instead, the system assigns servers to half of the unit interval, *i.e.*, all server mapped regions sum to half of the total unit interval. Half occupancy is maintained as an invariant to make sure that there is always an assignment of partitions satisfying the needs of all servers as well as an unassigned partition available to a recovered server. File sets that hash into un-mapped regions are re-hashed until assigned to a mapped region as shown in Figure 2. Re-hashing is performed using the next hash function among an agreed upon family of hash functions. On average, the system requires two probes to assign a file set, but we note that a hash probe does no I/O when determining where a file set is served. Successive hash probes incur negligible costs and happen with probability 2^{-r} after r rounds.

The system manages server and workload heterogeneity by changing the sizes of server's mapped regions. ANU randomization initially assigns servers mapped regions of equal length, because it has no knowledge of server capabilities. Each server monitors its performance and produces a performance metric over a chosen time interval. In this paper, we use latency as the performance metric – a natural choice as the metadata workload consists of little data and short-lived transactions. At the end of each interval, each server computes its latency in the past interval and reports it to an elected delegate server. The delegate server examines all latencies and comes up with an "average" value [40] for the whole system. The delegate scales down the mapped regions for servers above the average and scales up the mapped regions for servers below the average. The delegate is designed to be stateless and determines the new load configuration based solely on reported latencies. If the delegate fails, the next elected delegate runs the same protocol with the same information.

The system reorganizes load placement to conform to configuration changes made by the delegate. The delegate distributes a new mapping of servers to the unit interval to all servers. This is the only replicated state needed by our algorithm. Upon receiving updates to its mapped regions, a server identifies *shed* file sets – file sets that it served in the previous configuration that are served by another server in the current configuration. The shedding server flushes its cache with respect to shed file sets to create a consistent disk image. Then, the server hashes each shed file set to locate a new server and notifies the new server that it is gaining workload.

In addition to handling server and workload heterogeneity, scaling the server mapped regions deals gracefully with variance in hashing. For two identical servers A and B, we might expect them to have mapped regions of the same size. However, hashing variance may place more load at A than B initially. A reduces its mapped region by a large factor to shed more load. Mapped region scaling results in better load balance than simple randomization even when all servers and all file sets are homogeneous.

ANU randomization performs well when servers fail or recover, or when servers are installed or removed, maintaining good load balance and preserving load locality. When a server fails, the load it can take effectively goes to zero. Other servers increase their mapped regions to preserve the half-occupancy invariant of the unit interval. Only the file set(s) that were served previously by the failed server are re-hashed to locate a new server. When a server recovers or is added, it is assigned to a free partition and all other servers are scaled back to preserve the half-occupancy invariant. The framework treats commissioning (installing) or decommissioning servers the same as a recovery or failure respectively. However, if the added server increases k (the number of servers) such that there are fewer than $2^{\lceil \lg k \rceil + 1}$ partitions, the algorithm re-partitions the unit interval. The combination of the number of partitions and the half-occupancy invariant ensures that there is always an available partition into which a recovered or added server may be placed. We present an example in Figure 3. The system starts with 4 servers in 8 partitions with a highly skewed workload. The first server occupies almost all of four partitions, while the remaining three servers have little pieces in the remaining four partitions. Adding a fifth server re-partitions the unit interval, creating new partitions for more servers to be added. As long as

Figure 3. Partitioning the unit interval when adding a server.

each server occupies at most one single partial partition, free partitions are always available. Further partitioning the unit interval does not move any existing load and does not change the hash functions that address load, as does linear hashing [20]. The system moves the minimum amount of workload possible by scaling the mapped regions of servers from last configuration. Therefore, load locality is maintained and caches of file sets are preserved.

Load balance in this scheme is within a small constant factor of optimal. For n servers and m file sets, each server contains load $\lceil \frac{m}{n} + 1 \rceil$ with high probability. This result depends on several factors including a multiple choice heuristic that we have not described [5]. This variance is as small as any known bound for randomized placement and compares favorably to simple randomization in which load is bounded by $\lceil \frac{m}{n} + \Theta(\frac{\lg n}{\lg \lg n}) + 1 \rceil$.

5 Performance Evaluation

We evaluate the performance of our load management system based on ANU randomization against three other systems using a simulator driven by both a trace workload and synthetic workload. Simulation results verify that our system achieves load balance among heterogeneous server nodes, provides consistent performance for applications, and moves minimal amount of load when tuning load placement. Simulation results also indicate that simple randomization cannot cope with skew and heterogeneity and virtual processor systems require a larger shared state to achieve similar performance to our system.

5.1 Simulation Setup

For the evaluation, we constructed a trace-driven simulator using the YACSIM toolkit, which is a C-based library for discrete event simulation. The simulator models a shared-disk server cluster and servers use a first-in-first-out queuing discipline for workload.

In some previous experiments [40], we have used a one-hour DFSTrace [27] workload that contains 21 file sets and 112,590 requests to drive our simulation. Because DFSTrace has some deficiencies, such as being collected on legacy hardware with limited heterogeneity, we use a synthetic workload to drive most of the experiments in this paper. This allows us to experiment with an arbitrary amount of heterogeneity and helps to understand our system's characteristics under different hardware/workload configurations. We use DFSTrace workload results for comparison with synthetic workloads to ensure the sanity of our results.

Our experiments include both server heterogeneity and workload heterogeneity. Based on the number of file sets, we simulate a five-server cluster. Servers 0, 1, 2, 3, and 4 have processing power 1, 3, 5, 7, and 9 respectively to stress heterogeneity. More specifically, for the same workload, if the least powerful server in our simulated five-server cluster (server 0) consumes time T to complete a metadata request, then the most powerful server (server 4) consumes time $T/9$. In the synthetic workload, the total amount of workload in each file set is defined as Xc where X is randomly chosen from interval [1,10] and c is a scaling factor tuned to avoid overload of the whole system.

We investigate the performance characteristics of our system against three other load management systems: *simple randomization*, *dynamic prescient*, and *virtual processor*. Simple randomization employs a pseudo-random hash function to uniformly assign file sets to servers, allowing us to compare our system with static, offline randomized policies used in heterogeneous clusters. Dynamic prescient realizes the optimal load balance through identifying the permutation of file sets onto servers that minimizes average latency, because it has perfect knowledge of server capabilities and workload properties. It provides the upper bound of load balancing. The virtual processor system first randomly distributes file sets into Nv virtual processors where N is the number of physical servers and v is a scaling factor chosen from interval [1,10] to tune the total number of virtual processors in the system. By default, we set the value of v to be 5. The system then utilizes perfect knowledge about server capabilities and virtual processor workload characteristics to map virtual processors to servers in a way that minimizes average latency. This mapping procedure is similar to that in dynamic prescient except that the workload assignment and movement unit is now virtual processor instead of file set. In the experiments, we use two minutes as the load placement tuning interval for our system, the dynamic prescient system, and the virtual processor system in order to avoid over-tuning while still providing responsiveness. It is possible to update load placement at any time scale.

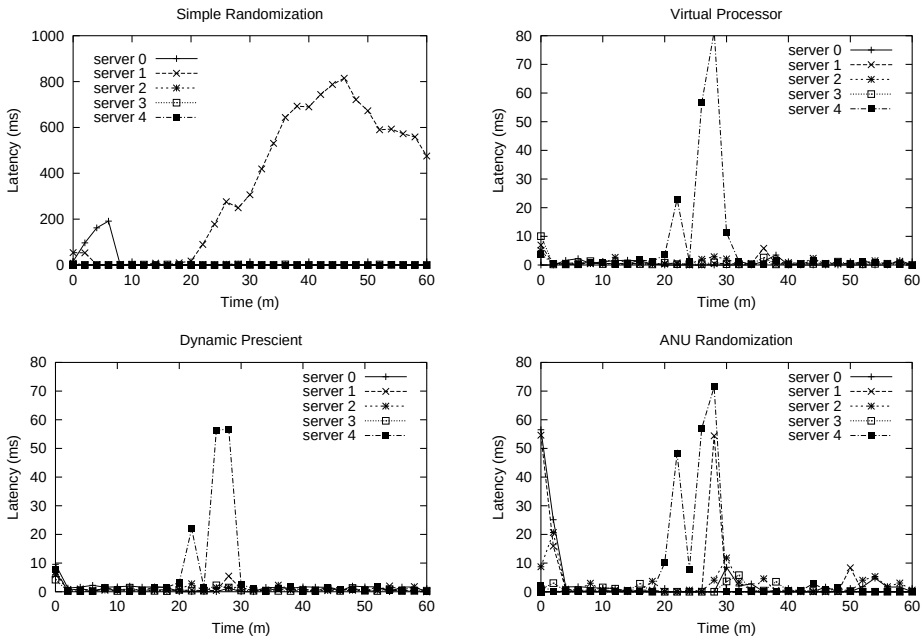


Figure 4. Server latency for file systems trace (DFSTrace) workloads.

5.2 General Comparison

The general performance comparison is done in two parts. First, we compare server latency in the four load management systems to verify our system achieves load balance. Second, we compare some aggregate metrics of our system with dynamic prescient and virtual processor systems to understand some load balance performance details, such as throughput and performance consistency.

5.2.1 Server Latency

Figure 5 shows the latency of the five servers when serving the synthetic workload over a two-hundred-minute interval. The synthetic workload consists of 66,401 requests against 50 file sets in a period of two hundred minutes. The request inter-arrival times in each file set are governed by a Pareto distribution that is heavy-tailed. As shown in Figure 5, simple randomization performs poorly. It is static algorithm and assumes homogeneity in server capabilities. Therefore, it cannot respond to skew in load placement. The weakest server's performance keeps degrading during the simulation and there is unused capacity on more powerful servers.

Having prescient knowledge of server capacities and workload characteristics, dynamic prescient and virtual processor keeps the system balanced from the very beginning, time 0. In the virtual processor system, the most powerful server (server 4) has higher latency, or in other words, lower performance than the second most powerful server (server 3) around time 150. This is due to the larger size of workload assignment unit and load skew among virtual processors, which makes it difficult to do fine-grained tuning and assign load proportional to each server's capacity. ANU randomization has no a-priori knowledge of server heterogeneity and workload characteristics. It initially assumes all servers and workloads are uniform. But it quickly adapts to heterogeneity and reaches load balances after several rounds of load placement tuning. The results in Figure 5 follow our findings in Figure 4 (presented in previous work [40]). Simulations based on file system traces indicate the sanity of our synthetic workload by showing the same scaling and tuning properties on real workloads.

5.2.2 Aggregate Metrics

Figure 6(a) shows the aggregate average latency of all requests in the synthetic workload and its standard deviation. Dynamic prescient has the best aggregate average latency, because it employs optimal load mapping that minimizes load skew. Therefore, dynamic prescient also provides an upper bound of aggregate average latency, or equivalently, throughput. Although virtual processors use prescient knowledge of heterogeneity to do load mapping as well, they perform slightly worse than dynamic prescient due to load skew from the large size of workload unit. The latency of ANU randomization is fairly close to that of dynamic prescient, indicating that ANU randomization provides throughput close to the upper bound without using a-priori knowledge of heterogeneity.

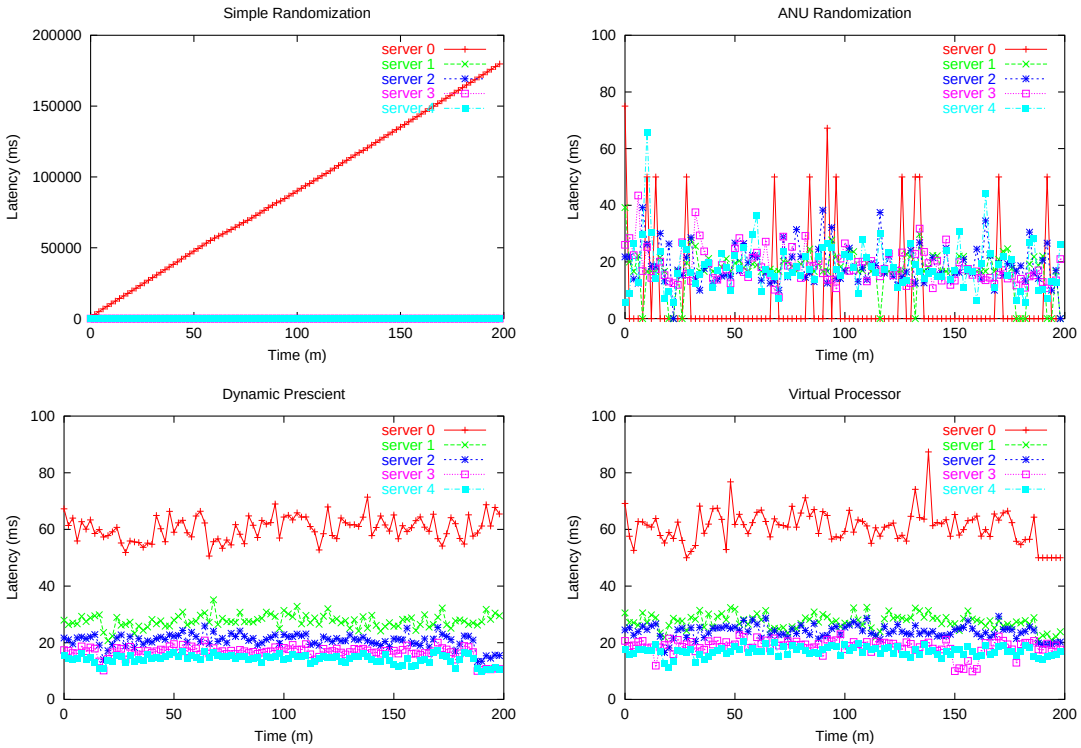
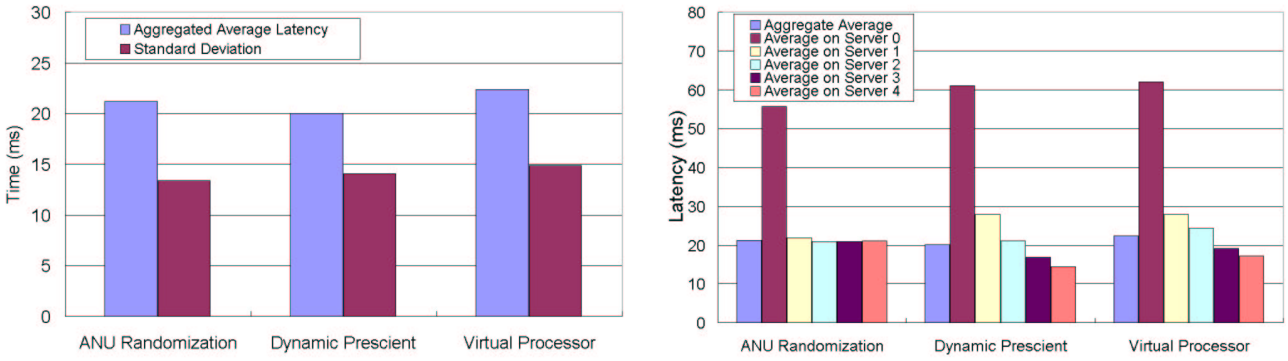


Figure 5. Server latency for synthetic workloads.



(a) Aggregate average latency and standard deviation.

(b) Average latency of requests served on each server.

Figure 6. Aggregated metrics comparison.

Figure 6(b) presents the average latency of tasks served by each individual server. Servers exhibit consistent average latency values in ANU randomization system, except sever 0, the weakest server. A thorough analysis of the experimental log files indicate that server 0 served only 248 requests (0.37%) out of the total 66,401 requests, and most of the 248 requests were served on server 0 before ANU randomization reached load balance. Therefore, the inconsistency of server 0 does not introduce significant skew into system-wide performance consistency. These observations indicate that application workloads will observe consistent latency over any non-idle server in the cluster once the system reaches balance. It will benefit applications that have strict performance requirements. Moreover, as cluster and grid systems extend to support Service Level Agreements [7, 8], it is essential that application performance

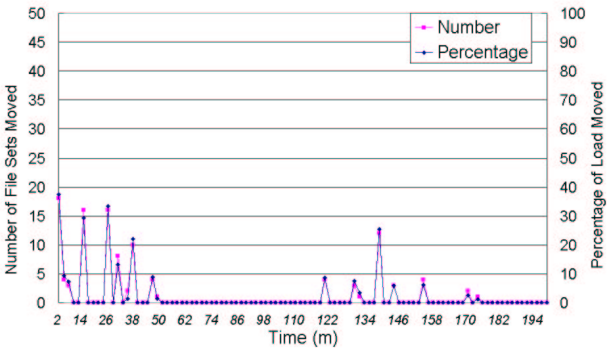


Figure 7. Load movement during the synthetic workload simulation.

is consistent over different servers in a heterogeneous cluster or even in a large-scale grid. Results from dynamic prescient and virtual processor are also shown in Figure 6 as references.

ANU randomization manages extreme server heterogeneity by ensuring the most powerful servers to achieve good load balance while allowing some extremely weak servers to sit idle, as revealed in Figure 5 where server 0 mostly sits idle after the system reaches load balance. ANU randomization identifies such incompetent components and notifies administrators.

Putting together results from the general comparison experiments show that our load management scheme performs comparably to the upper bound (dynamic prescient) and provides the load balancing benefits of virtual processor schemes. Considering the fact that our system achieves load balance without a-priori knowledge of heterogeneity, it provides a competitive approach to load management.

5.3 Variable Load Experiments

One of the design goals of our system is to minimize load movement when balancing load. It is very costly to move workload of a file set from one server to another in shared-disk clusters. The releasing server needs to flush its cache, writing all dirty data to stable storage. The acquiring server must initialize the file set. Furthermore, the acquiring server starts with a cold cache, which hinders initial performance. Therefore, our system is relatively conservative in moving load in response to short-term bursts in workload.

Figure 7 illustrates both the number of file sets moved by ANU randomization over the course of synthetic workload simulation and the percentage of total workload that has been moved during the same experiment.

Figure 7 shows that ANU randomization preserves load locality, or in other words minimizes load movement, during tuning. During the first several rounds of tuning, ANU randomization actively moves load among servers by scaling server's mapped regions to adapt to server and workload heterogeneity. During the whole simulation, which consists of 100 rounds of tuning, our system totally moves 112 file sets.

5.4 Comparison with Virtual Processor System

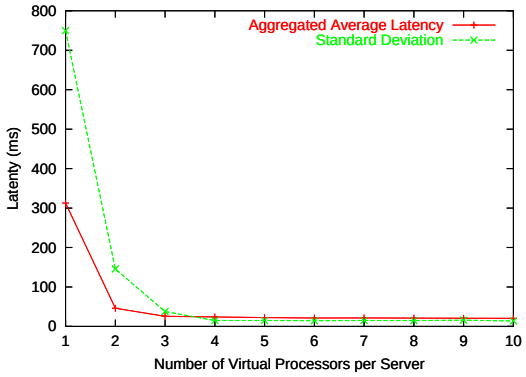
There is an important factor that affects the performance of virtual processor systems: number of virtual processors. With a small number of virtual processors, each will be assigned more workload, which increases the size of workload unit. However, with a large workload unit, it is difficult to perform fine-grained load tuning and assign to each server load proportional to its capacity, making load placement vulnerable to skew and imbalance. In contrast, a large number of virtual processors divides load finely, at the expense of increased state. Because virtual processor does not provide an efficient addressing scheme, it is essential to keep the address information for each individual virtual processor.[1] Considering large clusters consisting of tens of thousand of physical servers, maintaining the shared state for many more virtual processors becomes a serious concern.

On the other hand, the unit interval is the only shared state in ANU randomization system. Therefore, it scales with number of servers. The unit interval contains the information of each server's mapped region and provides an efficient addressing scheme. We hash any given file set's unique name into the unit interval to locate the server that is serving the file set, with possible re-hashing if the previous hash falls into un-mapped region.

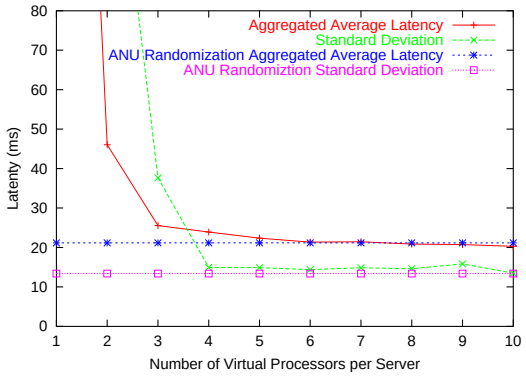
Figure 8(a) illustrates the tradeoff of virtual processor systems. We vary the number of virtual processors from 5 to 50, because we simulate 5 servers and 50 file sets. Each server can be assigned an arbitrary number of virtual processors. With a small number of virtual processors, the virtual processor system does not effectively balance the synthetic workload, yielding bad performance for applications. Load placement is greatly improved with a large number of virtual processors, which indicates a larger shared state.

Figure 8(b) shows a closeup of Figure 8(a) and directly compares ANU randomization with virtual processor systems. The virtual processor system achieves equivalent performance to ANU randomization when using 30 virtual processors for the 50 file sets, which indicates a relatively large shared state. When the number of virtual processors reaches 50, the virtual processor system outperforms ANU randomization on latency and performs comparably to the dynamic prescient system. When the number of virtual

[1]The addressing information could also be implemented in the Chord-style ring [35] to avoid replication at the expense of $\log(n)$ probes to the data structure.



(a) Performance of virtual processor system with different number of virutal processors.



(b) Performance of virutal processor system with different number of virtual processors (Enlarged).

Figure 8. Performance of virtual processor system.

processors is equal to the number of file sets, each virtual processor contains only one file set on average, which indicates the workload assignment and movement unit is effectively file set. The comparison in Figure 8 shows that virtual processor systems need to maintain a large shared state to achieve similar performance to ANU randomization.

6 Conclusion

This paper shows that our load management system based on ANU randomization technique addresses several performance issues in heterogeneous, shared-disk clusters. Experimental results indicate that ANU randomization deals with heterogeneity in both server and workload and performs comparably to a prescient system. The results also demonstrate that ANU randomization maintains performance consistency, minimizes load movement, and provides all the load balancing benefits of virtual processor systems with less shared state.

Our system has some properties that benefits cluster and grid systems, including scalability, efficient addressing, and load preservation. The scalability and addressing features of ANU randomization compare favorably to both bin-packing load balancing schemes [36, 43] and virtual processor schemes, in which any workload unit can be placed onto any server. To locate file sets, each computer must maintain a table that maps file sets to a particular server. This can represent a large amount of state to maintain and replicate when dealing with large numbers of file sets, increasing the time to reconfigure servers when moving load or recovering from a failure. For load balancing, ANU randomization also compares favorably to both bin-packing and virtual processor schemes. During load tuning and system configuration changes, ANU randomization moves a minimum number of file sets, preserving the cache contents of servers and reducing restart recovery times.

References

[1] K. Amiri, D. Petrou, G. R. Ganger, and G. A. Gibson. Dynamic function placement for data-intensive cluster computing. In *Proceedings of the 2000 USENIX Annual Technical Conference*, 2000.

[2] D. Anderson, J. Chase, and A. Vahdat. Interposed request routing for scalable network storage. In *Proceedings of the Symposium on Operating Systems Design and Implementation*, 2000.

[3] L. Aversa and A. Bestavros. Load balancing a cluster of web servers using distributed packet rewriting. In *Proceedings of the IEEE International Performance, Computing, and Communications Conference*, 2000.

[4] A. Barak, G. Shai, and R. G. Wheeler. *The MOSIX Distributed Operating System: Load Balancing for UNIX*. Springer Verlag, 1993.

[5] A. Brinkmann, K. Salzwedel, and C. Scheideler. Compact, adaptive placement strategies for non-uniform capacities. In *Proceedings of Symposium on Parallel Algorithms and Architectures*, 2002.

[6] V. Cardellini, M. Colajanni, and P. Yu. Geographic load balancing for scalable distributed web systems. In *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2000.

[7] J. Chase, L. Grit, D. Irwin, J. Moore, and S. Sprenkle. Dynamic virtual clusters in a grid site manager. In *Proceedings of the International Symposium on High-Performance Distributed Computing*, 2003.

[8] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems, 2002.

[9] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, 2001.

[10] D. L. Eager, E. D. Lazowska, and J. Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engineering*, 12(5), 1986.

[11] W. Feng, M. Warren, and E. Weigle. Honey, I Shrunk the Beowulf! In *Proceedings of the International Conference on Parallel Processing*, 2002.

[12] G. R. Ganger, J. D. Strunk, and A. J. Klosterman. Self-* storage:

Brick-based storage with automated administration. Technical Report CMU-CS-03-178, Carnegie Mellon University, 2003.

[13] G. R. Ganger, B. L. Worthington, R. Y. Hou, and Y. N. Patt. Disk subsystem load balancing: Disk striping vs. conventional data placement. In *Proceedings of the International Conference on System Sciences*, 1993.

[14] M. Harchol-Balter and A. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems*, 15(3), 1997.

[15] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1), 1988.

[16] IBM. Total Storage SAN File System, 2003. http://www.storage.ibm.com/software/virtualization/sfs/.

[17] L. V. Kale, M. Bhandarkar, and R. Brunner. Run-time support for adaptive load balancing. In *Proceedings of the 4th Workshop on Runtime Systems for Parallel Programming*, 2000.

[18] E. D. Katz, M. Butler, and R. McGrath. A scalable HTTP server: The NCSA prototype. *Computer Networks and ISDN Systems*, 27(2), 1994.

[19] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[20] W. Litwin, M. Neimat, and D. A. Schneider. LH* — a scalable, distributed data structure. *ACM Transactions on Database Systems*, 21(4), 1996.

[21] C. Lu and S. Lau. An adaptive load balancing algorithm for heterogeneous distributed systems with multiple task classes. In *Proceedings of the International Conference on Distributed Computing Systems*, 1996.

[22] Lustre: A scalable, High-Performance file system. Technical Report available at – http://www.lustre.org/docs/whitepaper.pdf, Cluster File Systems, 2002.

[23] J. Menon, D. A. Pease, R. Rees, L. Duyanovich, and B. Hillsberg. IBM storage tank: A heterogeneous scalable SAN file system. *IBM Systems Journal*, 42(2), 2003.

[24] M. Mitzenmacher. On the analysis of randomized load balancing schemes. In *the ACM Symposium on Parallel Algorithms and Architectures*, 1997.

[25] M. Mitzenmacher. How useful is old information? *IEEE Transactions on Parallel and Distributed Systems*, 11(1), 2000.

[26] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10), 2001.

[27] L. Mummert and M. Satyanarayanan. Long term distributed file reference tracing: Implementation and experience. *Software - Practice and Experience*, 26(6), 1996.

[28] N. Nedeljkovic and M. J. Quinn. Data-parallel programming on a network of heterogeneous workstations. In *Proceedings of the First International Symposium on High Performance Distributed Computing*, 1992.

[29] Panasas. Shared Storage Cluster Computing, Oct 19 2003.

[30] S. Petri and H. Langendorfer. Load balancing and fault tolerance in workstation clusters migrating groups of communicating processes. *ACM SIGOPS Operating Systems Review*, 29(4), 1995.

[31] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in structured P2P systems. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, 2003.

[32] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM 2001*, 2001.

[33] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, 2001.

[34] P. Shenoy and H. Vin. Efficient striping techniques for multimedia file servers. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video*, 1997.

[35] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM*, 2001.

[36] J. Watts, M. Rieffel, and S. Taylor. Dynamic management of heterogenous resources. In *Proceeding of the High Performance Computing Conference: Grand Challenges in Computer Simulation*, 1998.

[37] J. Watts and S. Taylor. A practical approach to dynamic load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 9(3), 1998.

[38] J. Wilkes. Data services - from data to containers, 2003. Keynote at the Usenix Conference for File and Storage Technologies.

[39] M. Willebeek-LeMair and A. P. Reeves. Strategies for dynamic load balancing on highly parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(9), 1993.

[40] C. Wu and R. Burns. Handling heterogeneity in shared-disk file systems. In *Proceedings of the International Conference for High Performance Computing and Communications*, 2003.

[41] C. Wu and F. Lau. *Load Balancing in Parallel Computers: Theory and Practice.* Kluwer Academic Publishers, Boston, Mass, 1997.

[42] S. Zhou. A trace-driven simulation study of dynamic load balancing. *IEEE Transactions on Software Engineering*, 14(9), 1988.

[43] S. Zhou, J. Wang, X. Zheng, and P. Delisle. Utopia: A load-sharing facility for large heterogeneous distributed computing systems. *Software – Practice and Experience*, 23(12), 1993.

[44] H. Zhu, T. Yang, Q. Zheng, D. Watson, O. H. Ibarra, and T. R. Smith. Adaptive load sharing for clustered digital library servers. *International Journal on Digital Libraries*, 2(4), 2000.