To Appear in MobCops Workshop of the 2nd IEEE Collaborative Computing Conference (CollaborateCom 2006), Atlanta, GA, USA, Nov 2006

A Collaborative Approach to Stochastic Load Balancing with Networked Queues of Autonomous Service Clusters

Cheng-Jia Lai, and Wolfgang Polak
FX Palo Alto Laboratory
3400 Hillview Ave, Building 4
Palo Alto, CA 94304, United States

***Abstract*—Load balancing has been an increasingly important issue for handling computational intensive tasks in a distributed system such as in Grid and cluster computing. In such systems, multiple server instances are installed for handling requests from client applications, and each request (or task) typically needs to stay in a queue before an available server is assigned to process it. In this paper, we propose a high-performance queueing method for implementing a shared queue for collaborative clusters of servers. Each cluster of servers maintains a local queue and queues of different clusters are networked to form a unified (or shared) queue that may dispatch tasks to all available servers. We propose a new randomized algorithm for forwarding requests in an overcrowded local queue to a networked queue based on load information of the local and neighboring clusters. The algorithm achieves both load balancing and locality awareness.**

I. Introduction

Load balancing has been an increasingly important issue for handling computational intensive tasks in a distributed system using Grid [1][2] and cluster computing. There are several reasons for the increasing use of such distributed systems since the 1990s. First, by deploying parallel algorithms, scientists can achieve complex computing tasks on workstations and personal computers as fast as on a supercomputer at a fraction of the cost. Second, the fast growth of the Internet, wireless networks, and mobile computing has resulted in many applications that require a distributed computing environment due to the dispersed locations of the data and users. Last but not least, as the number of users and requests can increase exponentially in a wide area network such as the Internet (e.g., for a period after the 9/11 event, traffic to CNN's web site doubled every 7 minutes [27]), servers can become heavily loaded within a short period of time, creating 'hot-spots' with vastly prolonged waiting times of user requests in the queue [6][10].

Thus, in Grid and cluster computing environments, multiple instances of a server, sometimes called a server pool or a cluster, are installed for handling a class of user requests. Any request for a computing task will be queued until one available server is assigned to process it. If no server in this cluster is available, forwarding it to another cluster is possible but networking those clusters can be difficult in terms of efficiency, scalability, and security. As of 2004, there had been 956 Condor [17] pools (clusters of host computers of servers) involving 37,790 hosts globally. Search engine provider Google reportedly acquired about 100,000 computers in its own compute and data clusters [25][16]. As recorded [26] in June 2006, Grid MP Global has contained 3.6 million computing devices and more than 1.3 million members world-wide.

In general, with Queueing Theory [6] it has been shown that a shared queue for multiple servers can outperform separate queues each for one server since the shared queue allows any idle server to be assigned a task whenever the shared queue is nonempty, i.e. a request is waiting. However, it is practically very difficult to implement such a shared queue when those servers are installed in geographically dispersed locations on the network, i.e. in separate clusters, especially for a global company of which the intranet covers multiple sites on the earth. For example, Condor pools use flocking and gliding [18] in order to find an available server outside a cluster but with limited scalability and performance in a global network.

Thus, in this paper, we propose a high-performance queueing method to simulate a shared queue for collaborative clusters of servers. Each cluster of servers maintains a local queue. The queues of individual clusters are networked to form a "shared queue" for all servers in those clusters so that a task can be forwarded from the queue of its originating cluster to a remote queue when the local queue is overcrowded, to reduce the waiting time of this task with a small cost of network delay in the transmission. We present a new method for forwarding tasks to remote queues using a randomized message-exchange algorithm between neighboring queues, without using any centralized control.

II. Networked Queues

Referring to Figure 1, consider that multiple instances of a service are installed in clusters where each cluster has a queue to hold the incoming requests into the cluster. If there is still some server idle available in the cluster, one of the idle servers will immediately be assigned to handle this request.

Otherwise, a decision will be made whether to queue the request locally or to forward it remote cluster. The local queue will make the forward-or-not decision for every single request entering the queue. The decision making process relies on a randomized algorithm that takes as input the current states of the local queue as well as the latest known states of the neighboring queues.

Specifically, the probability P_k that the queue in cluster k accepts an incoming request is described as a function of N_k and m_k, where N_k is the queue length of cluster k, and m_k is the number of servers in cluster k. Note that other ways to assign P_k are possible, e.g. based on expected service time, etc. We assume that each cluster is autonomous at deciding (and reporting) its probability of accepting new requests.

For simplicity, throughout our analysis, we just define P_k as

$$P_k = \frac{m_k}{N_k + m_k}$$

such that P_k is 1 when the local queue is empty and decreases when the local queue grows.

The networked queues maintain links between each pair of neighboring queues. Such a link is created when two clusters have mutually agreed to support each other as first priority when the other has suddenly received a very high number of requests in its queue. Note that those queues that are not directly linked can still share their servers indirectly since it is possible for a request to be forwarded more than once, i.e. over multiple hops to a remote queue. For each cluster k let Φ_k be the set of neighbors.

Instead of implementing a 'perfect' shared queue, our approach creates a framework for cooperating networked queues that collectively perform similar to a shared queue.

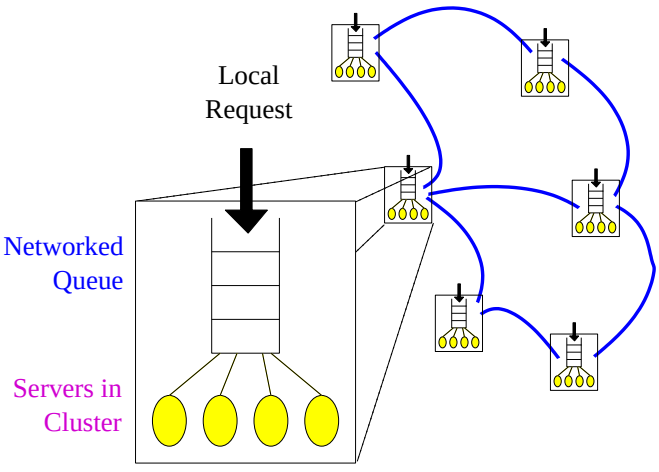

Figure 1 – Networked queues for clusters of services

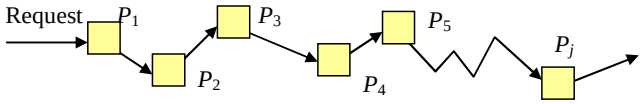

Figure 2 – Stochastic acceptance in forwarding a request between queues

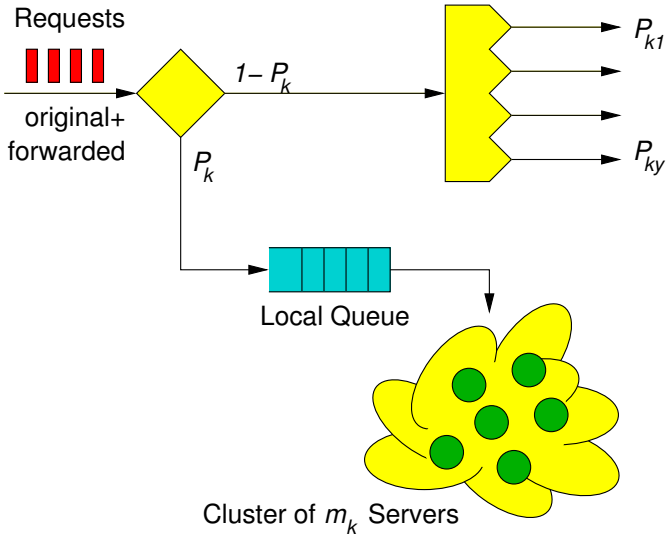

Figure 3 – The decision process of accepting or forwarding an incoming request

Referring to Figure 2, on the forwarding path of a request where the request can be forwarded more than once, the j-th visited queue independently determines if it accepts this request by storing it, or instead forwards it again to the next queue.

Referring to Figure 3, the decision process is comprised of two stages. First, with probability P_k, the queue will simply accept the request. Otherwise, the request will be forwarded to a random neighbor y selected with probability P_{ky}. In order to determine P_{ky}, each queue, say, of cluster k, maintains a local state (P_k, R_k) where P_k is as previously defined, and, using the convention $P = 1 - \overline{P}$ for probabilities, R_k is defined as

$$\overline{R}_k = \overline{P}_k \cdot \prod_{\forall y \in \Phi_k} \overline{P}_y .$$

Intuitively, R_k is the probability that either k or one of its neighbors will accept a new request. In addition, each queue retains the state of each of its neighbors; neighboring queues periodically exchange their local state.

When the queue of cluster k decides to forward a request, the request is forwarded to a neighbor y with probability P_{ky} where

$$P_{ky} = C \cdot \left(1 - \frac{\overline{R}_y}{\overline{P}_k}\right),$$

where R_y is the value in the latest y's state that this queue k has known, and C is a scale factor chosen such that

$$\sum_{y \in \Phi_k} P_{ky} = 1.$$

In practice, a request may be rejected by a local queue due to reasons other than the current load, for example, due to a failure of remote user authentication or billing negotiation. These in effect make P_k smaller. We do not consider these extra mechanisms in this paper.

III. Simulation Results

We evaluate the proposed queueing method using simulations. We chose to use simpler scenarios with which we can observe the performance of the queueing method for preliminary evaluation. Extensive studies on more specific and complex scenarios by simulations will be reported in future research. This section describes some of our initial results.

Figure 4 describes three different topologies used in our simulations with 5, 10, and 20 clusters, respectively. A vertex represents a cluster, and an arc between two vertices represents the bidirectional neighbor relationship between two clusters. Clusters exchange state information (message exchange) and forward requests along arcs. We assume that in both cases the network delay is 35 ms. Every 50 ms, each cluster sends a message to update all neighbors of the current state of its P and R variables. Upon receiving such a message from a neighbor, a cluster re-computes its local value of R but will not update its neighbors until the next periodic message exchange.

Each cluster consists of a pool of 5 servers and one local first-in-first-out queue. Each server obtains its tasks from the queue and each server is capable of processing any request. Delivering a request from the queue to a server involves 5 ms of network delays and 1 ms of processing time.

Each cluster has a private user population where users arrive and depart at random times. An active user will generate a random sequence of requests for that cluster. Several distributions define the generation of requests:

- The inter-arrival times between users are a Poisson process; the arrival rate varies based on the traffic scenario of the simulation (see below).
- Each user stays in the private user population only for a duration of which the length is randomly chosen from a Poisson distribution with the average equal to 0.5 second.
- The intervals between requests from a single user are a Poisson process with a rate chosen between 0 and 20/sec at random with even distribution. Therefore, every user generates 5 requests on average.
- For simplicity, we assume that every request requires 40 ms to be processed by any server, with no need of remote data access.

For each request, the *system time* is defined to be the duration from the time it is generated by the user to the time it finishes being processed by a server. For example, if a request is generated at simulation time 5.5 (seconds) and a server in some (possibly remote) cluster finishes processing it at time 6.8, its system time of the request is 1.3 seconds. Note that we do not consider the time to deliver any possible results back to the user.

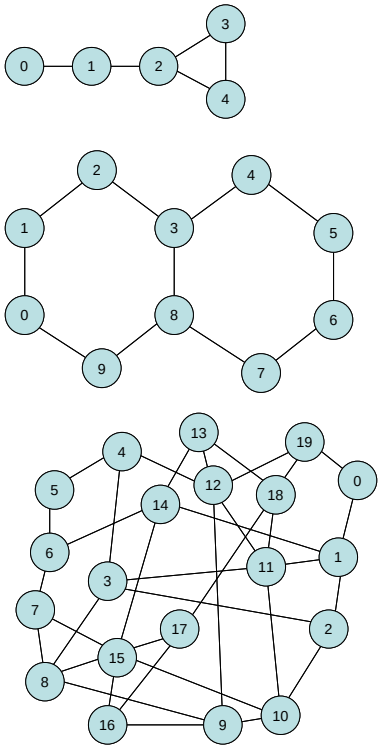

Figure 4 – Topologies of networked queues with 5 (top), 10 (middle) and 20 (bottom) clusters in simulations

Referring to Figure 5, we run two simulations both with the 5-cluster topology in Figure 4 and the same random sequence of user populations and generated requests, but one simulation deploys isolated queues that do not forward requests, while the other deploys the networked queues in our proposal. The user arrival rates in all clusters are 10/sec (therefore, on average 50 requests per second) beginning at simulation time 0.3, except that the user arrival rate to 50/sec (therefore, on average 250 requests per second) in cluster k at $2.3 + 5k$ sec for 2 seconds. It shows that all isolated queues suffer from the burstiness of user arrivals that temporarily exceeds the processing capacity of a cluster (125 requests per second); the numbers of queued requests sharply increase to hundreds and remain significantly high for about 10 seconds, which lasts much longer than the 2-second duration of the peak user-arrival rate. In contrast, the networked queues show

significantly lower queue length that return to normal very quickly after the user load decreases.

Figure 6 shows the same traffic load and networked queue length in relation to the local acceptance probability P_k (and R_k) in the different clusters. Due to the sharp increase of N_k and forwarding of excess user requests the acceptance probabilities decrease not only in the loaded cluster but also in neighboring ones. It shows that stochastic load balancing can benefit from asynchronous peak user request rates in different clusters, i.e., available servers in near clusters, not necessarily immediately neighboring, may be used to process excessive local requests. In fact, stochastic load balancing is advantageous if not all the peak user-request durations in clusters overlap.

Figure 7 shows the system times of user requests by different user populations, with the average, maximum, minimum, and standard deviation (std dev) values, from 0 sec to 45 sec. To better show the distribution of the system times, we draw a bar for the range [minimum, average + std dev] since we have found that in all user populations, (average – std dev) is lower than minimum. For comparison, we also calculate a "speedup" factor of the system times for using networked queues instead of isolated queues. The speedup of the average system times in a user population is the ratio of the average system times using isolated queues to the average system times using networked queues. The speedups of the maximum, minimum, and std dev system times are calculated similarly. As shown, the speedups are significant except that the speedup of the minimum system times is always one because in these cases the local queues are nearly empty and load balance is not needed.

We find that the speedups of the average system times are in in the range of 4 to 6. In addition, the speedups (actual a reduction) of the standard deviations of system times are even larger than the speedups of the average system times. Thus, the stochastic load balancing provides much more predictable system times against the stochastic arrivals of user requests. Another major speedup is on the maximum system times. The maximum value shows the longest time a request has needed to be processed in a user population. This is very important for real-time systems and human-interactive applications.

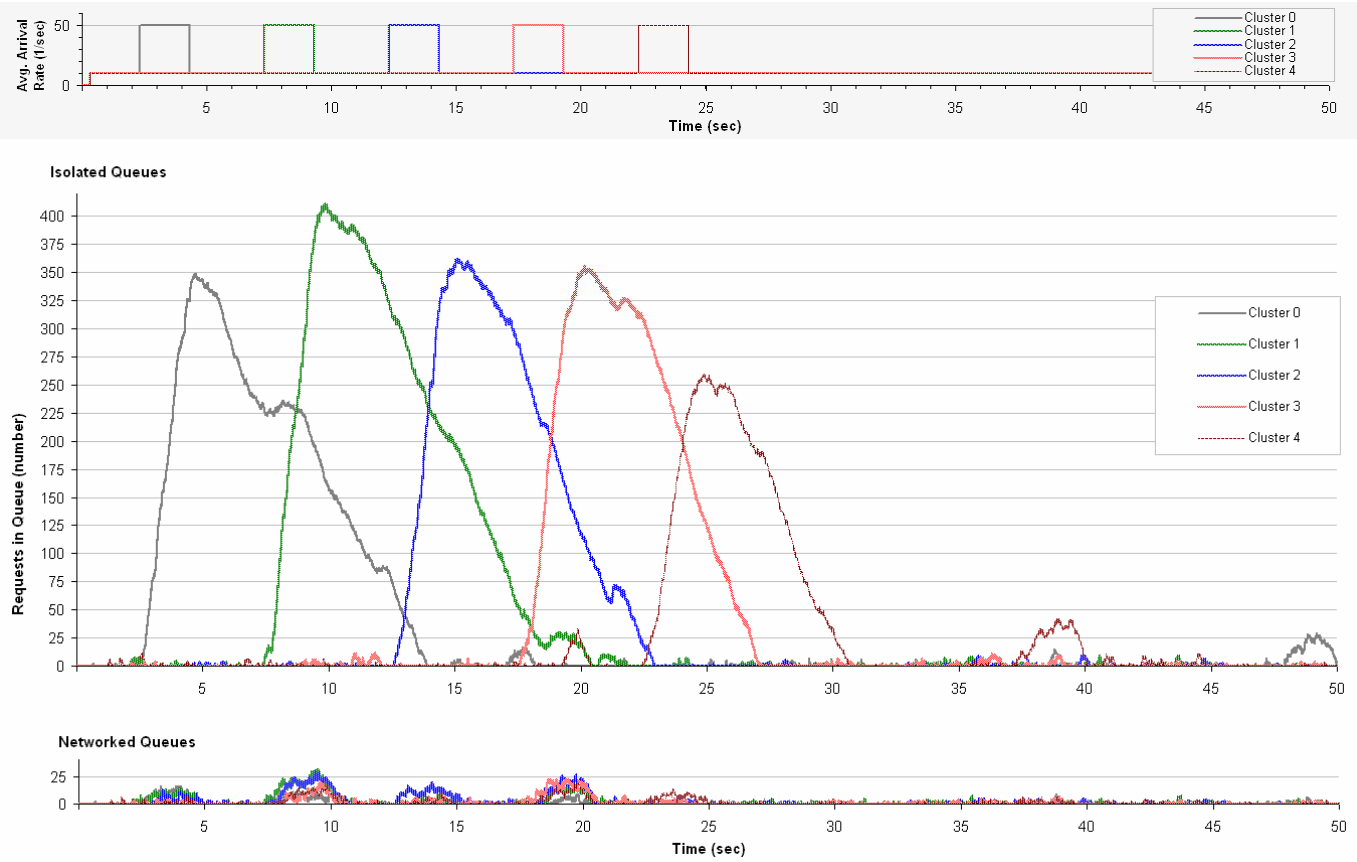

Figure 5 – The upper diagram shows the inter-arrival rates of user populations in different clusters changing over the time, and the lower two aligned diagrams show the comparison of isolated queues (top) and networked queues (bottom). It shows that the collaborative queueing method can significantly reduce the number of queued requests seen by an incoming request, which may result in much shorter system times stochastically with better load balance.

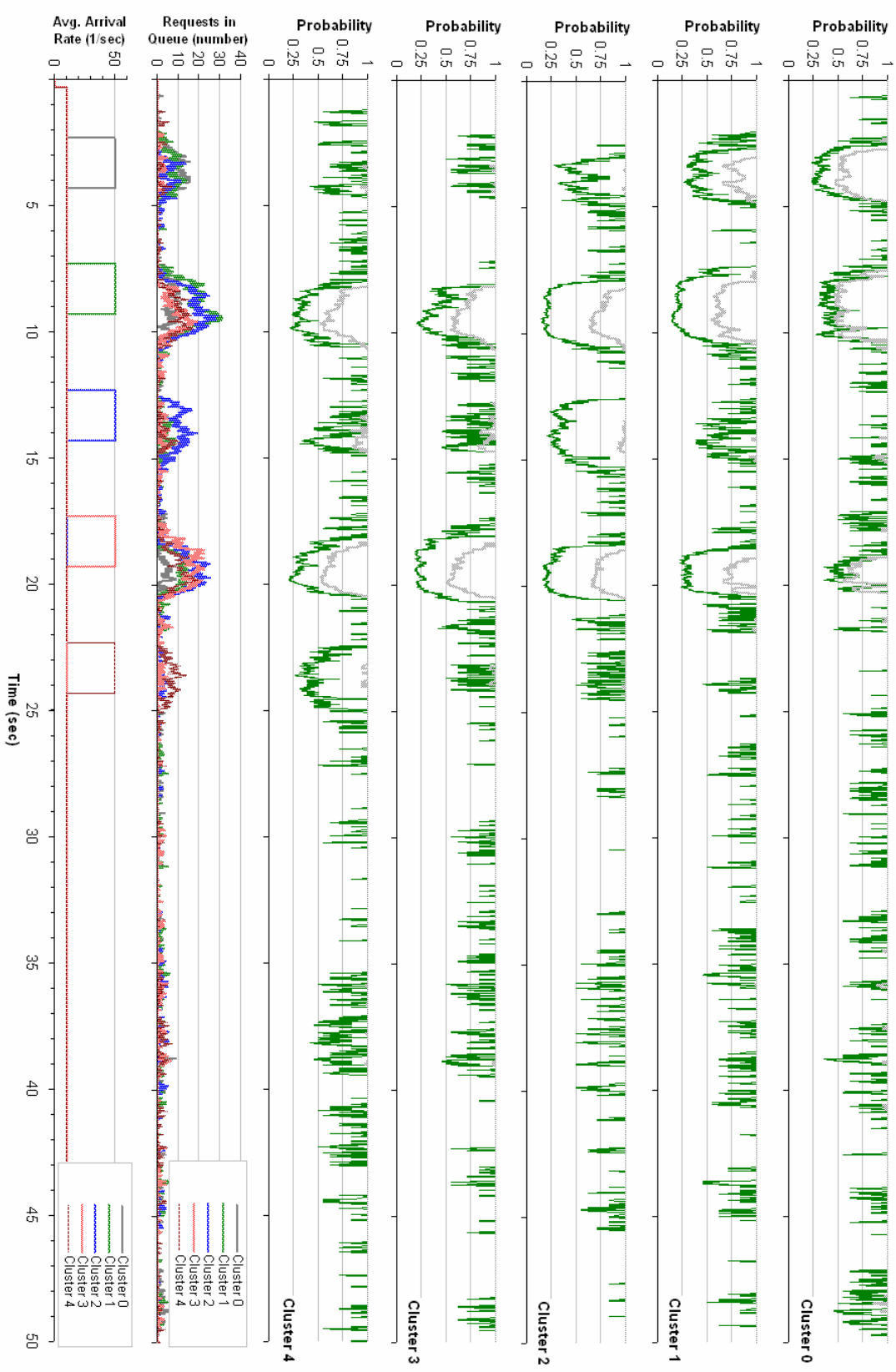

Figure 6 –The right five diagrams show the P_k (dark) and R_k (light) values at each cluster k along the time with networked queues in the five-cluster topology in Figure 4. The left two diagrams are simply copied from Figure 5.

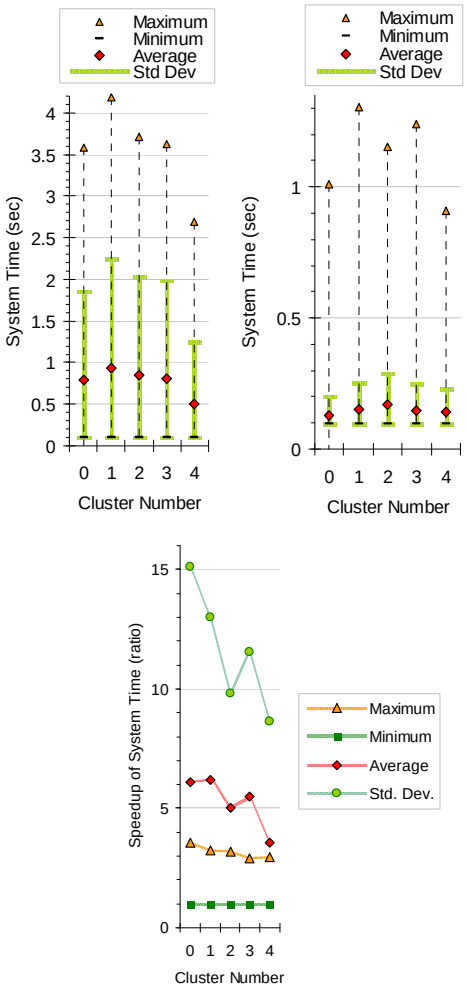

Figure 7 – The system times of user requests in simulation with 5 clusters, measured in the first 45 seconds. The top-left diagram shows the case with isolated queues, and the top-right diagram shows the case with networked queues. The bottom diagram shows the speedup factors of the system times using networked queues instead of isolated queues, which is considered an evidence of stochastic load balance.

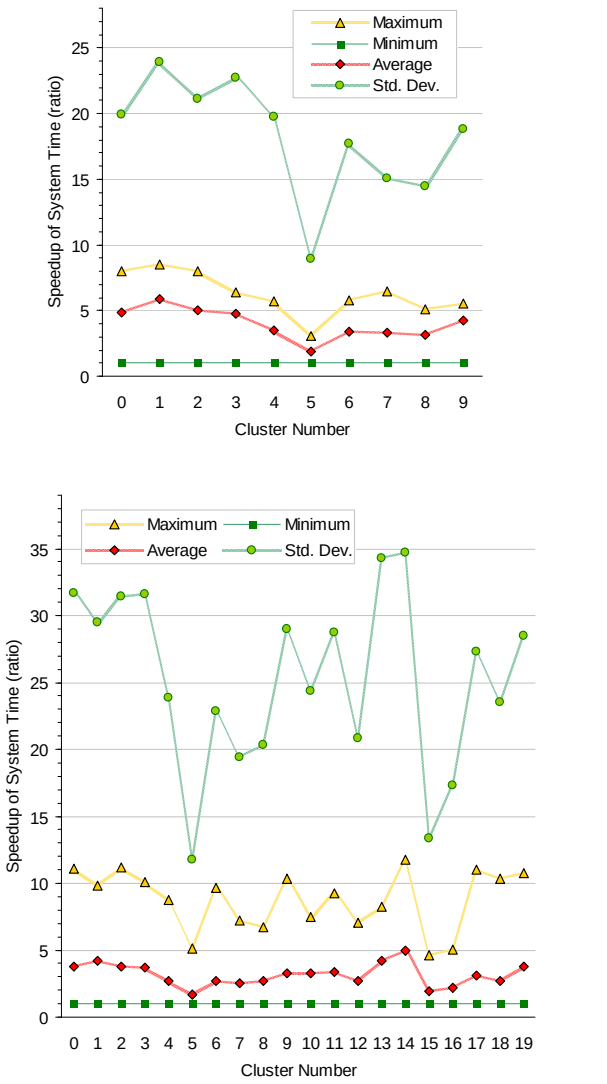

Figure 8 – The speedups of the system times in simulations with 10 (top) and 20 (bottom) clusters, measured in the first 70 and 120 seconds, respectively.

Figure 8 is the speedup statistics of the system times measured in the simulations with 10 and 20 clusters, respectively. The user arrival rates in all clusters are 10/sec since 0.3 sec except that the user arrival rate rises to 50/sec in cluster k at $2.3 + 5k$ sec for 2 seconds. Since with more clusters it takes a longer time to include durations of peak user arrivals in all clusters, the average, maximum, minimum and std dev of the system times are calculated in the first 70 ($= 5\times10 + 20$) seconds for the 10 clusters, and in the first 120 ($= 5\times20 + 20$) seconds for the 20 clusters, resulting in the smaller average and std dev values than those in the 5-cluster simulation. Thus, although the speedups of the average system times do not look as large, it does not mean that the speedups are less significant, that is, the speedups of the average system times are still essentially high. In fact, the speedups (reduction) of std dev increase sharply in most of the user populations, regardless of the effect of a longer measurement duration. The speedups of the maximum system times are also larger with more clusters.

Figure 9 is the statistics of the numbers of times a request has been forwarded before it is accepted to enter some queue, with the 20-cluster topology, in the first 120 seconds. We find that the maximum numbers of forwarding are seven, and most of the requests are accepted without being forwarded, into the local queue where it originates. Thus, our approach prefers the local queue to a remote queue, when the local queue is lightly loaded, i.e. has a high acceptance probability, P, with a small N, which implies that the approach is locality aware. Typically, locality-awareness is desirable since it can limit the overhead of remote data access and/or data replication, and perhaps also

the monetary costs due to business agreements on processing a remote task.

IV. Discussion

Each cluster of servers maintains a local queue, and those queues are networked to form a unified (or shared) queue for all those servers in clusters. The method for forwarding requests in an overcrowded queue to some other queues uses a randomized algorithm with carefully designed probabilities according to relative queue lengths. It comes along with a periodic message-exchange algorithm between the networked queues to represent priorities of support between clusters. It allows the local queues to be networked as mutual agreements of support, which makes the grid of services adaptive to business partnerships and scalable with the number of participants.

The proposed queueing method can forward request between queues with locality-awareness, based on neighborhoods, and load-balance, based on state information of neighboring clusters. At the same time it allows each cluster to maintain a local queue for fast processing of locally originating requests, with the low overhead typical in a server farm. The local coordination and decision making achieves global load balance and can help to implement a large-scale service grid across different sites (or subsidiaries) of a global company. Computational intensive tasks can be performed without the need to co-locate a significant number of servers on one site, but multiple servers on different sites can collaborate for supercomputing or grid computing. Business contracts (or mutual agreements of support) are considered as the links between queues.

The algorithm can use measures of load other than the relative queue length. For instance, P_k can be based on the expected time to complete a request in cluster k. Similarly, it is possible to consider resources other than compute cycles. The forwarding of request may involve significant amounts of data transfer and the probabilities for forwarding to a particular neighbor may be adjusted to account for data size and available bandwidth. Other resources to be considered include databases that may only be available or cached at certain locations and specialized hardware needed to service the request.

For example, a service that can process video indexing can be installed with multiple instances at various locations across the intranet of a company. Then, the load balance and locality awareness of sharing these service instances may provide high performance of the video indexing service to the public and/or to the internal users, reusing the already invested computers. In fact, since the utilization of computers is time-variant and dynamic to user behaviors, a global intranet could have a lot of its computers idle in one region while others busy in others so it is beneficial to share those computing capacities across the intranet.

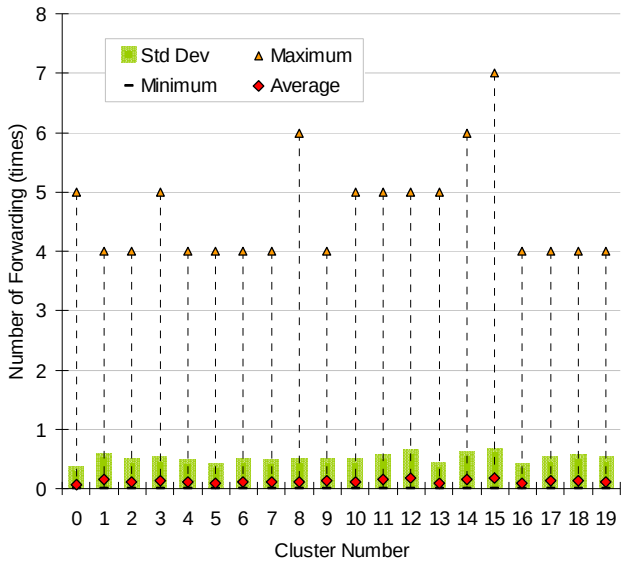

Figure 9 – The statistics of the numbers of times a request has been forwarded before it is accepted to enter a queue, categorized by the private user populations, in the simulation with 20 clusters (see Figure 4)

V. Conclusion

We have proposed a new queueing method for collaborative clusters of servers where each cluster maintains a local queue while requests can be forwarded between those queues. Each cluster has its autonomy with the decisions of accepting or rejecting a new incoming request, according to its own states and rules

The collaborative queueing method does not require global knowledge of those queue states, but instead, uses only states exchange between neighboring queues, to provide a scalable and timely mechanism. The neighbor relation can change dynamically and the algorithm will adapt to the new topology. Our method allows requests to be forwarded to a queue that is likely to be shorter and that typically results in a shorter waiting time. Preference is given to queues nearer the request's origin.

For example, it allows the local queues of companies and organizations to collaborate with others by some mutual agreements of support. It creates a grid of services and resources that is flexible, adapts to business partnerships, and scales with the number of participants. The local control aspects of the algorithm make is possible to operate across firewalls as long as a single neighbor link bridges each boundary, forming a peer-to-peer (P2P) overlay network.

In these features our algorithm is different from other schemes for distributed load balancing, most of which require global control of some form (e.g. a central scheduler) or require users to make reservations or decide on server allocations. We have not yet done an extensive performance comparison with global allocation schemes since these make

fundamentally different assumptions and are not directly comparable.

REFERENCES

[1] I. Foster, and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1998.

[2] I. Foster, and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *International Journal of Supercomputer Applications and High Performance Computing*, 1996.

[3] K. Czajkowski, I. Foster, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A resource management architecture for metacomputing systems," *Lecture Notes in Computer Science*, 1997.

[4] S. Ogura, S. Matsuoka, and H. Nakada, "Evaluation of the inter-cluster data transfer on Grid environment," in *Proceedings of the 3rd IEEE/ACM Int'l Symposium on Cluster Computing and the Grid*, 2003.

[5] S.-M. Park, and J.H. Kim, "Chameleon: A resource scheduler in a data grid environment," in *Proceedings of the 3rd IEEE/ACM Int'l Symposium on Cluster Computing and the Grid*, 2003.

[6] L. Kleinrock, *Queueing Systems*, Volume I & II, Wiley, New York, 1975.

[7] P. Davie, *Computer Networks: A System Approach*, Morgan Kaufmann, San Francisco, 2000.

[8] D. Bertsekas, and R. Gallager, *Data Networks*, 2nd Ed. Prentice Hall, New Jersey, 1992.

[9] S. Keshave, *An Engineering Approach to Computer Network*. Addison-Wesley, 1997.

[10] A. Ravindran, D. Phillips, and J. J. Solberg, *Operations Research: Principles and Practice*, 2nd Ed. John Wiley & Sons, Inc., 1987.

[11] T. Johnson, "Designing a distributed queue," in *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing*, pp. 304–11, San Antonio, Texas, October 1995.

[12] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design*, 2nd Ed. Addison-Wesley, 1994.

[13] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, B. Mancheck, and V. Sundderam. *PVM: Parallel Virtual Machine—A User's Guide and Tutorial for Network Parallel Computing*, MIT Press, 1994.

[14] S. Djilaili, "P2P-RPC: Programming scientific applications on peer-to-peer systems with remote procedure call," in *Proceedings of the 3rd IEEE/ACM Int'l Symposium on Cluster Computing and the Grid*, 2003.

[15] R. H. Arpaci-Dusseau. *Performance Availability for Networks of Workstations*. PhD thesis, University of California, Berkeley, 1999.

[16] S. Ghenmawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proceedings of the 19th ACM Symposium on Operating System Principles*, 2003.

[17] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: The Condor experience," in *Concurrency: Practice Experience*, 2004; 0:0-20, John Wiley & Sons, Ltd., 2004.

[18] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne, "A worldwide flock of Condors: load sharing among workstation clusters," in *Journal on Future Generations of Computing Systems*, Vol. 12, 1996.

[19] B.C. Neuman, and S. Rao, "The Prospero resource manager: A scalable resource reservation framework for scientific computing," in *Scientific Computing in Object-Oriented Parallel Environments*, pp. 283-290, Springer-Verlag, 1997.

[20] J. Weissman, "Gallop: The benefits of wide-area computing for parallel processing," Technical Report, Univ. of Texas, San Antonio, 1997.

[21] A. Grimshaw, W. Wulf, J. Frnech, A. Weaver, and P. Reynolds, Jr., "Legion: The next logical step toward a nationwide virtual computer," Technical report CS-94-21, Dept. of Computer Science, University of Virginia, 1994.

[22] L, Rudolph, M Slivkin-Allalouf, and E. Upfal, "A simple load balancing scheme for task allocation in parallel machines," in *Proceedings of the 3rd annual ACM symposium on Parallel algorithms and architectures*, pp. 237 - 245, 1991.

[23] V. Cardellini, M. Colajanni, and P. S. Yu, "Dynamic load balancing on web-server systems," in *IEEE Internet Computing*, May-June 1999.

[24] K. Shen, T. Yang, and L. Chu, "Cluster load balancing for fine-grain network services," in *Proceedings of Int'l Parallel and Distributed Processing Symposium (IPDPS 2002)*, 2002.

[25] J. Dean, and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proceedings of USENIX OSDI 2004*, 2004.

[26] Grid MP Global Statistics, at http://www.grid.org/stats/.

[27] W. LeFebvre, "CNN.com: Facing a world crisis," in *Proceedings of the 15th Systems Administration Conference (LISA 2001)*, San Diego, California, Dec 2-7, 2001.