01 Jan 2005

Measuring Scalability of Resource Management Systems

A. Mitra

Muthucumaru Maheswaran

Shoukat Ali
Missouri University of Science and Technology

Recommended Citation

A. Mitra et al., "Measuring Scalability of Resource Management Systems," *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, Institute of Electrical and Electronics Engineers (IEEE), Jan 2005.
The definitive version is available at https://doi.org/10.1109/IPDPS.2005.277

Measuring Scalability of Resource Management Systems

Arindam Mitra
Dept. of Computer Science
University of Manitoba
Winnipeg, MB R3T 2N2
Canada
arindam@cs.umanitoba.ca

Muthucumaru Maheswaran
School of Computer Science
McGill University
Montreal, QC H3A 2A7
Canada
maheswar@cs.mcgill.ca

Shoukat Ali
Dept. of Elec. and Comp. Engg.
University of Missouri-Rolla
Rolla, MO 65409-0040
USA
shoukat@ece.edu

Abstract

Scalability refers to the extent of configuration modifications over which a system continues to be economically deployable. Until now, scalability of resource management systems (RMSs) has been examined implicitly by studying different performance measures of the RMS designs for different parameters. However, a framework is yet to be developed for quantitatively evaluating scalability to unambiguously examine the trade-offs among the different RMS designs. In this paper, we present a methodology to study scalability of RMSs based on overhead cost estimation. First, we present a performance model for a managed *distributed system (e.g., Grid computing system) that separates the manager and managee. Second, based on the performance model we present a metric used to quantify the scalability of a RMS. Third, simulations are used to apply the proposed scalability metric to selected RMSs from the literature. The results show that the proposed metric is useful in quantifying the scalabilities of the RMSs.*

1. Introduction

The essence of modern distributed computing systems (like Grids) lie in the maximal utilization of virtual resources within a large federated system [4, 8]. A key component required in this respect is the *resource management system* (RMS), that orchestrates the resource allocation activities in the federated system so that appropriate resources are assigned to a given application task [11]. Consequently, efficient performance of the distributed system largely depends on the scalability of its RMS.

In distributed computing, until recently [14], scalability was only investigated qualitatively. In this approach, a distributed system is considered scalable if no performance bottlenecks exist in the system (i.e., the operational overhead is evenly distributed across the system). In contrast, in the framework proposed by [14], scalability of a distributed system is quantitatively studied using a formal process that provides a scalability metric. Their scalability metric tracks the variation of throughput per unit overhead for increasing system sizes. Because the framework presented in [14] is targeted towards measuring the overall scalability of a *complete* distributed system, it is less suitable for measuring the scalability of a portion of a distributed system. For instance, the overall "productivity" can decrease if a particular component is unscalable, even when all the remaining ones are scalable. In such situations, it becomes harder to determine the extent of performance degradation due to a given component, e.g., the RMS. Instead, a component by component analysis of a distributed system will be better, as it will allow the "scalability bottleneck" to be isolated.

As a first cut at this problem, we consider two logical components for distributed systems: a manager (RMS) and a managee (set of resources), and examine the scalability of the manager with respect to the managee. We consider a given RMS configuration as scalable, if it can be tuned with minimal cost to handle larger managee configurations while maintaining a certain level of overall efficiency. The RMS tuning is performed by adjusting a predefined set of "scaling enablers" that is specific to the particular RMS.

A scalability measurement framework such as ours can be used in several different ways such as: (a) the ability to design RMSs that can handle "scaled up" configurations of the deployment time managee without any decrease in overall efficiency and (b) the ability to analyze the re-usability of an existing RMS design as the overall system is reorganized.

Section 2 presents the scalability model and a metric for evaluating the scalability of a managed distributed system. Section 3 applies the scalability metric to example systems using simulations. The results indicate that the proposed metric is able to estimate the scalability of the target systems. In Section 4, we examine the related works.

Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)
1530-2075/05 $ 20.00 **IEEE**

2. Scalability Model and Metric

2.1. The Distributed System Model

Here, we discuss the distributed system model used in this paper. In this model, distributed systems are categorized as *managed* and *unmanaged* systems. In managed distributed systems, the operations are explicitly coordinated to maximize some measure of an overall performance measure of the system. A managed system can dedicate part of its components as coordinators that orchestrate the activities or distribute the coordination among all its components. Examples of practical systems that fall into this class include Grid computing systems [4, 8], application hosting centers [10], and cluster-based servers [21, 19]. In unmanaged systems, the operations of a system are not explicitly coordinated to maximize a performance measure of the system. Instead, unmanaged systems perform local operations which if properly designed, can maximize some measure of delivered performance. An example of an unmanaged distributed system is the peer-to-peer (P2P) system [13]. In P2P systems, an emergent behavior can result from the searches that are performed with localized information.

With the manager/managee division, the manager needs to maintain status information about the managee to coordinate the activities within the managee. Maintaining state about the managee incurs cost due to a variety of factors such as dissemination of status updates and processing of status updates. In addition to maintaining state, the manager also needs to perform computations to make optimized scheduling decisions.

It should be noted that useful work (work delivered to the clients) is performed by the managee. Therefore, the work consumed by the manager is essentially overhead. Ideally, we want to incur zero overhead for maximum possible useful work out of a particular configuration of the system. In real systems, the output of the managee is dependent on the manager because it orchestrates the activities within the managee.

2.2. Scalability Model

A key component of a scaling model of a distributed system is the *scaling strategy* that determines how the system can be scaled up or down from a *base* configuration [14]. The scaling strategy achieves this by defining the scaling variables $x(k)$ for each *scale factor* k that dictate the growth and shrinkage of the system. As an example consider a compute cluster that scales up from $k = 1$ to $k = 2$. Let the scaling strategy $x(k)$ be a vector of two variables, where $x_1(k)$ is the number of servers and $x_2(k)$ is interconnection network bandwidth. At scale factor $k = 1$, $x_1(k) = 10$ and $x_2(k) = 10$Mbps and at $k = 2$, $x_2(k) = 100$ and $x_2(k) = 100$Mbps.

Once the configuration of the distributed system is changed by the scaling variables, the new configuration should be fine tuned by adjusting a set of scaling enablers $y(k)$ to operate in the most optimal manner. For a given system, the set of scaling enablers $y(k)$ may be dependent on the choice of scaling variables $x(k)$. In the example compute cluster, $y(k)$ may be parameters such as routing algorithms and TCP window sizes. Different configurations of the distributed system can be obtained by changing the scale factor values. In the space defined by the scaling variables, we can represent these changes as a *scaling path* that defines the evolution of the distributed system.

The above scaling process is valid for unmanaged distributed systems. For managed distributed systems, the above process is modified so that only the managee (or the manager) is explicitly scaled by the scaling variables. The manager is then tuned by adjusting its set of scaling enablers so that it can adequately handle the scaled up system components. Tuning the RMS may increase the overhead caused by the RMS although it improves its capability. An RMS is considered scalable if the rate of increase of the overhead function with the scale factor is low. Based on the above discussion, we present the following definition for the scalability of an RMS.

Definition: Consider a managed distributed system, where an RMS is managing a resource pool (RP). Assume that the function $G(k)$ gives the minimum cost of maintaining RMS to manage the resource pool at scale k. Then, the scalability of the RMS at scale k is measured by the slope of $G(k)$.

2.3. An Isoefficiency Scalability Metric

Here, the definition of scalability is used to derive a scalability metric. We refer to this scalability metric as the *isoeffiency* scalability metric because it keeps the overall efficiency constant at a predefined value as in parallel systems [1].

For a given managed distributed system at scale factor k, let the useful work done by the system be $F(k)$, the overhead caused by the RMS be $G(k)$, the overhead caused by the RP be $H(k)$. The overall efficiency of the system is defined as useful work divided by total work. Hence, the efficiency of the managed system at scale k is given by:

$$E(k) = \frac{F(k)}{F(k) + G(k) + H(k)}$$

Various factors such as processing and communication overheads associated with state estimation and processing overhead associated with decision making con-

tribute towards the RMS overhead term $G(k)$. For the most part, the RP overhead term $H(k)$ is determined by the job control overheads and data management overheads.

Suppose k_0 denotes the scale of the base configuration, W denotes the useful work done by the system at scale k_0 (i.e., $W = F(k_0)$), O_{RMS} denotes the RMS overhead at scale k_0 (i.e., $O_{RMS} = G(k_0)$) and, O_{RP} denotes the RP overhead at base scale (i.e., $O_{RP} = H(k_0)$). Let $f(k)$ denote the the value of $F(k)$ normalized with respect to $F(k_0)$, or W. Let $g(k)$ and $h(k)$ be similarly normalized values of $G(k)$ and $H(k)$ respectively, such that: $f(k) = \frac{F(k)}{F(k_0)}$, $g(k) = \frac{G(k)}{G(k_0)}$, and $h(k) = \frac{H(k)}{H(k_0)}$. Then, for the base system, the efficiency is given by:

$$\begin{aligned} E(k_0) &= \frac{F(k_0)}{F(k_0) + G(k_0) + H(k_0)} \\ &= \frac{W}{W + O_{RMS} + O_{RP}} \end{aligned}$$

and for the scaled system (at scale factor k), using the above expressions, the efficiency $E(k)$ is given by:

$$E(k) = \frac{f(k)W}{f(k)W + g(k)O_{RMS} + h(k)O_{RP}}$$

From the isoefficiency requirement, the efficiency for the scaled configuration should be maintained at a desired value such that $0 < E(k_0) < 1$ and $E(k) = E(k_0)$. Letting $E(k_0) = \frac{1}{\alpha}$ we get:

$$\begin{aligned} \frac{1}{\alpha} &= \frac{f(k)W}{f(k)W + g(k)O_{RMS} + h(k)O_{RP}} \\ \alpha f(k)W &= f(k)W + g(k)O_{RMS} + h(k)O_{RP} \\ (\alpha - 1)f(k)W &= g(k)O_{RMS} + h(k)O_{RP} \\ f(k) &= c \cdot g(k) + c' \cdot h(k) \qquad (1) \end{aligned}$$

where, c and c' are $constants$. One of the underlying assumptions in scaling up a base configuration of the managed distributed system is that the RP is scalable (i.e., the RP does not become a bottleneck). But the RP will always incur some non-zero cost. Then from Equation (1) we can say that:

$$f(k) > c \cdot g(k) \qquad (2)$$

Equation (2) implies that the useful work performed by the system should grow at least at the same rate as RMS overhead to keep efficiency constant. Section 3 provide examples of applying the scalability metric and associated conditions to example RMSs.

3. Scalability Analysis of Example Systems

3.1. Simulation Setup

The simulator is written using Parsec [15]. The network topologies used by the simulator are extractions from the Mercator topology generator [16]. To these topologies, we map elements such as routers, schedulers, and resources to obtain Grid topologies. We assume homogeneous resources and schedulers with finite processing and storage capacity. Similarly, network links have finite bandwidth and non-zero latencies. The simulator uses an OSPF [9] like algorithm for routing messages between resources. The set of resources are separated into non-overlapping clusters and each cluster is coordinated by a scheduler. The scheduler receives status updates from resources either periodically or on a need basis.

We use synthetic workloads based on parallel moldable workloads of supercomputing environments [22, 23]. A job in these workloads is characterized by arrival instant, partition size (number of processors used by the job), execution time, requested time as an upper bound to a job's execution time, and job cancellation possibility. In this paper, we assume a partition size of 1 and zero job cancellation possibility.

Arriving jobs are classified as: (a) LOCAL: jobs that need to execute locally or closer to the point of submission and (b) REMOTE: jobs that are suitable for remote execution. Because we do not model data transfers among jobs, the only constraint dictating local execution is job size, i.e., jobs with execution time lower than T_{CPU} are LOCAL. A job execution will be considered successful if it completes within its expected completion time specified by a user "benefit" function U_b. Table 1 shows the values used by T_{CPU} and U_b.

3.2. Procedure for Scalability Analysis

Figure 1 shows a flowchart for the scalability measurement process. Two important parts of the process are determining the best scaling path for the RP or the RMS and determining the minimal modification costs of the RMS. Because this paper does not examine data transfers between jobs, determining a feasible scaling path for the RP is straightforward and is not considered here. Selecting the set of scaling enablers such that efficiency remains constant for minimum cost is an optimization problem for which we use a simulated annealing [2, 12, 5] procedure.

The following are the major steps in measuring the scalability of an RMS:

Step 1: Choose a feasible value for the efficiency E_0 that should be kept constant as the system scales.

Step 2: Scale the RMS or the RP by selecting the best scaling path. When scaling the RP, a simulated annealing type of search can be used for this search. If a scalable RP cannot be found, then the base system is considered unscalable.

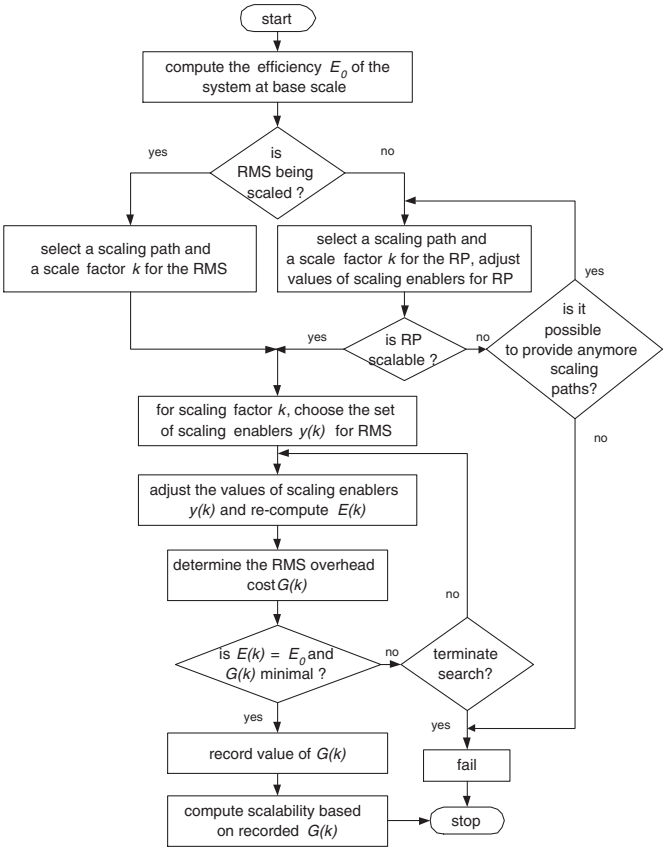

Figure 1. Flowchart for the scalability measurement process.

Step 3: Tune the RMS using the scaling enablers to keep the overall efficiency at the selected value. A simulated annealing search is used to determine the set of scaling enablers such that overhead $G(k)$ is minimum at scale factor k.

Step 4: Compute the the scalability of the RMS based on $G(k)$.

3.3. Example Systems

We use seven RMS models from [6, 17, 24] to illustrate the applicability of our scalability measurement procedure. We implement all RMS models on our Grid model, which does not completely match the "native" models used in the above papers. This explains the differences between the results obtained in the original papers and the ones reported here. Besides CENTRAL other systems are based on the above papers.

CENTRAL: Here a centralized scheduler makes decisions for all the resources in the system. The resources update the scheduler every τ seconds with their loading conditions. If loading conditions at the resource did not change significantly from the previous update, an update might be suppressed. This update optimization is used by all periodic updating schemes.

LOWEST: The RMS consists of multiple schedulers with each receiving periodic updates from non-overlapping clusters of resources [17]. On a LOCAL job arrival, a scheduler will schedule it on the least loaded resource it its cluster. On a REMOTE job arrival, a scheduler will poll a set of randomly selected L_p remote schedulers. The job is transferred for execution to a remote scheduler with the least loaded resources.

RESERVE: PULL type RMS. Here the schedulers are arranged as in LOWEST [17]. When average cluster load for a local cluster for a scheduler S_a falls below threshold T_l, then S_a advertises to register reservations at L_p remote schedulers. On a REMOTE job arrival, a scheduler will examine the average load of its local cluster. If it is above T_l, it probes the remote scheduler that made the most recent reservation. The job is sent to the remote scheduler if the loading there is below a given threshold. Otherwise, the reservations are cancelled.

AUCTION: When a new job arrives, a scheduler follows the same process as in LOWEST for initial scheduling [24]. When a scheduler S_a finds a resource in its cluster is idle or has load below threshold T_l, it sends out auction invitations to L_p neighboring schedulers. A scheduler S_b receiving the invitation finds a resource in its local cluster with load above T_l , it replies back with a bid to S_a. The auctioning scheduler S_a accumulates bids over a small interval and selects the bid from the bidder with the highest load.

S-I: PUSH type RMS. This sender-initiated strategy is somewhat similar to LOWEST [6]. Here, a set of autonomous local scheduler communicate with each other through a Grid middleware. We restrict each cluster to have single scheduler and model the Grid middleware using a simple queue with infinite capacity and finite but small service time. The scheduling strategies R-I and Sy-I also share this model.

On a REMOTE job arrival, a scheduler polls L_p remote schedulers. The remote schedulers respond with *approximate waiting time* (AWT), *expected run time* (ERT) for the particular job and *resource utilization status* (RUS) for the resources in their cluster. Based on the collected information, the polling scheduler calculates the potential *turnaround cost* (TC) at local cluster and each remote cluster. To compute the optimal TC, first the minimum approximate turnaround time ATT

is calculated as sum of the AWT and ERT. If the minimum ATT is within a small tolerance ψ for multiple schedulers, the scheduler with smallest RUS is chosen to accept the job.

R-I: Periodically, a scheduler S_x checks RUS for the resources in its cluster [6]. If the RUS for a resource in its cluster is below threshold δ, S_x decides to execute remote jobs and informs at most L_p remote schedulers. A remote scheduler S_y, receiving S_x's intention will send S_x the resource demands for the first job in its wait queue. When S_x replies back with its ATT and RUS, S_y uses this information to compute TC at local and remote sites and schedule the job accordingly.

Sy-I: This combines S-I and R-I [6]. As in R-I, each scheduler will advertise its own underutilized resources periodically. Based on this information a scheduler with a new job will schedule the job locally or send to to the advertising scheduler. However, if a new job arrives at a scheduler which has received no advertisements, it will use the S-I approach to schedule the job.

3.4. Simulation Results and Discussion

Here we present the results of the scalability experiments on the example systems. The efficiency $E(k_0)$ was kept in $[0.38, 0.42]$. To determine the minimum RMS overhead $G(k)$ for each configuration, we used a simulated annealing procedure to tune the scalability enablers of the RMS. Tables 2 - 5 show the combinations of scaling variables and scaling enablers used for the different experimental cases. For all experiments the workload was scaled in the same proportion as the scaling variable. The $G(k)$ values in this paper are defined as the overall time spent by the schedulers for scheduling, receiving, and processing updates. Table 1 shows the values for the common variables used for all the cases.

The slope in Figure 2 shows the rate of increase in RMS overhead as a system is scaled up as per the details in Table 2. A decreasing slope means that the RMS needs to do less work to sustain the system efficiently at the new scale k, compared to the last scale $k-1$. At base scale, $k=1$, the distributed models all incur substantially large overhead than the CENTRAL model. But as the systems are scaled up, the slope values for distributed models indicates that their RMS overhead is decreasing and therefore are scaling well. In contrast, the increasing slope for the CENTRAL model indicates rapid increase in RMS overhead. So clearly, the distributed models are more scalable for $1 < k \leq 6$, than the centralized model when scaling by network size. This agrees with intuition. Furthermore, the LOWEST is the most scalable distributed RMS while Sy-I is least scalable.

In Figure 3, the scaling factor is the rate of job completion by the RP, as per Table 3. The network size is fixed.

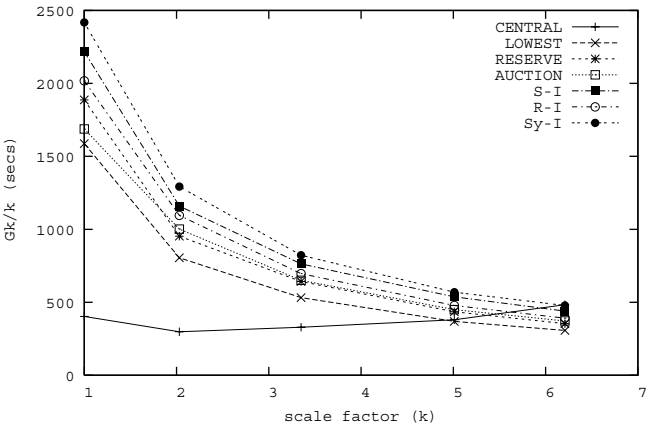

Figure 2. Variation in $G(k)$ on scaling the RP by number of nodes.

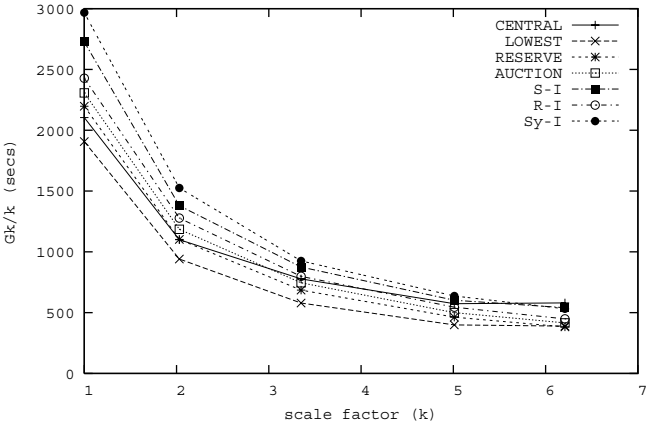

Figure 3. Variation in $G(k)$ on scaling the RP by service rate. Network size is 1000 nodes.

In this case, the CENTRAL model proves to be more scalable than majority of the distributed models at scaling factor $k \in [1, 3]$. However, at higher scales CENTRAL's overhead keeps on increasing and at $k = 6$ it is the least scalable RMS. Once again, LOWEST proves to be the most scalable among all models.

In Figure 4 and 5, only the RMS is scaled. The RP is unaltered. This is done to find a suitable RMS configuration to yield the best results for a given RP. Figure 4 is from experiment detailed in Table 4. It shows that as the number of estimators are scaled up, slope for models like AUCTION and Sy-I are no longer scalable after $k > 3$. These models use both PUSH and PULL technique for status estima-

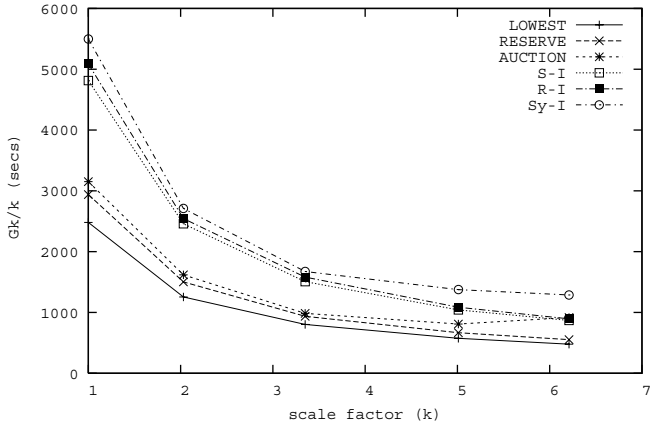

Figure 4. Variation of $G(k)$ on scaling the RMS by number of estimators. Estimators are the RMS nodes which receive the status updates from RP resources and distribute to the scheduling decision makers. Network size is 1000 nodes.

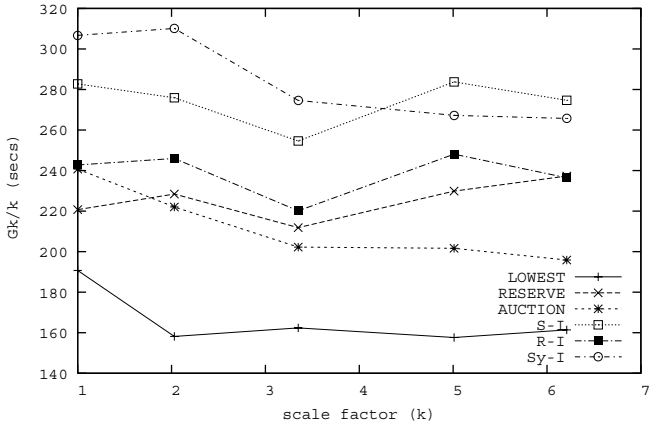

Figure 5. Variation in $G(k)$ on scaling the RMS by L_p (the number of neighbors being probed or polled). Network size is 1000 nodes.

tion and scheduling. The result shows that these RMSs are scalable only within a small range of scale factors. The results are further corroborated by their throughput and job response times as in Figure 6 and 7 respectively. Figure 6 shows that at higher scales throughput (number of jobs completed per unit time) for AUCTION starts falling after $k = 5$ and throughput Sy-I shows no improvement at $k > 4$. Similar results can be seen for job response times for these RMS models in 7.

In Figure 5, the scale factor is the number of neighbors each scheduler probes for scheduling REMOTE jobs, as per Tables 5. In this case, the PULL RMS models (LOWEST and S-I) shows slight improvement in scalability at $k = 2$. But at higher scales, $k > 2$, they incur high overhead and are no longer scalable. The PUSH RMS models (RESERVE and R-I), shows no consistent performances. However, RESERVE is clearly unscalable at $k > 3$. The RMSs using both PUSH and PULL techniques are scalable after $k > 2$.

4. Related Work

We classify the existing scalability measurement approaches into two classes: *qualitative* and *quantitative*. The qualitative approach fosters scalability by instituting a "rule of thumbs" for the design process, without presenting a formal methodology for scalability evaluation. Conversely, quantitative approaches provide metrics and formal methodologies for measuring the scalability of a target system. These approaches can be further divided as *indirect* and *direct*. In indirect approaches, scalability is examined by measurements based on traditional performance metrics such as throughput and response time. As these are widely studied in various contexts of distributed systems, we do not discuss them in this paper. In direct approaches, specially devised scalability metrics are used in the measurements.

One example of the qualitative approach is [3]. Here, the topology aggregation strategy is used to make *quality of service* (QoS) based routing more scalable. The topology aggregation is a popular strategy to reduce the overhead of status dissemination in routing problems. It improves the scalability by reducing the overhead while maintaining high performance levels.

Quantitative approaches are widely studied in the area of parallel computing systems [1, 18, 20, 7]. In [18], three different models for measuring speedup metrics are presented for homogeneous distributed memory multiprocessor systems. The models presented include: fixed-size speedup, fixed-time speedup and memory-bounded speedup. This work highlights the differences among the different scalability metrics. An isoefficiency function for a parallel system is defined in [1]. The isoefficiency function is based on growth in workload that is needed to keep the overall system efficiency constant as the number of processors grow. A survey of a number of scalability metrics is presented in [20]. Here the authors summarize the different situations for which different scalability measuring strategies are suitable.

Our work belongs to the quantitative-direct category.

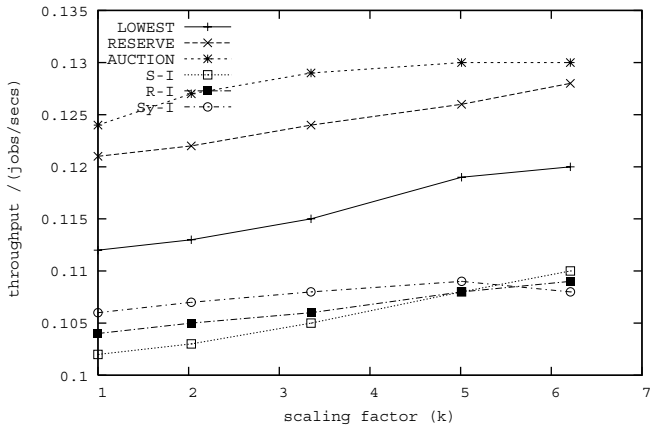

Figure 6. Throughput obtained by scaling RMS by number of estimators. Network size is 1000 nodes.

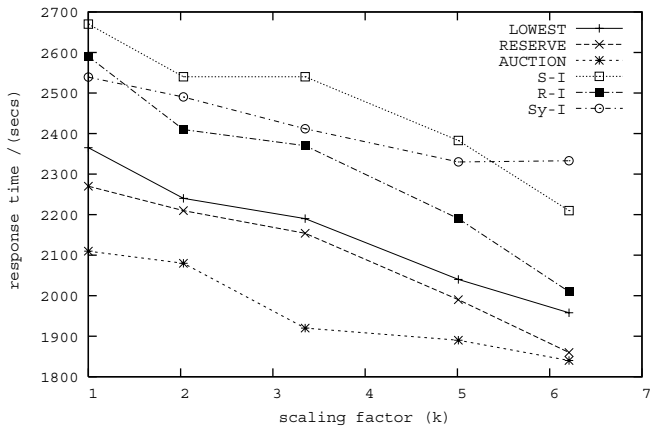

Figure 7. Average response times obtained by scaling RMS by number of estimators. Network size is 1000 nodes.

Though our approach draws heavily on the substantial work already done in parallel computing, among existing research that is applicable to generalized distributed systems, only [14] falls into this category. As mentioned before, our work differs from [14] because, in general, our approach allows measuring the scalability of the RMS in a distributed system. The scaling metric used by [14] is a function of the throughput obtained from the system, whereas our proposed metric is a function of the overhead obtained from the system while keeping the system efficiency constant. The scaling strategy used in our model is similar to that used in [14]. However, our framework is much more flexible and capable of incorporating a much wider range of scaling variables and also investigate the relevance of such variables in different situations (e.g., as in Case 3 and 4).

5. Conclusions and Future Work

This paper proposes a scalability measurement framework for RMSs. As part of this work, we presented a definition of scalability in the context of RMSs for managed distributed systems. From the definition we derived a metric for measuring the scalability of a given RMS. In addition, we derived certain conditions and a scalability measurement process for RMSs. We applied the proposed scalability metric and scalability measurement process to seven example RMSs. This helped to illustrate the operation of our proposed framework as well as its utility.

From the simulation results, we can observe that the proposed framework helps the scalability analysis in two different ways: (a) it determines whether a candidate scaling variable is indeed a feasible scaling variable, (b) what is the relative scalability of the different schemes along a given scaling strategy and (c) identify the scaling path (combination of scaling variables and scaling enablers) over which the system functions profitably. Once we select a definite set of scaling variables, we can perform the analysis to identify the relative scalability of different RMS schemes. Identifying the feasible scaling dimensions can be useful in a practical setting because we can confine the "tuning knobs" in a practical environment to these scaling variables to obtain the best performance from the RMS.

There are several topics that need further investigation including: (a) developing strategies to apply this framework to complex RMS architectures, (b) evaluating scenarios where jobs have data dependencies and precedence constraints among them and use the framework to measure the scalability based on the RP overhead $H(k)$, in this paper we have assumed $H(k)$ to be negligible.

References

[1] A. Grama, A. Gupta, and V. Kumar. Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures. *IEEE Parallel and Distributed Technology*, 1(3):12–21, Aug. 1993.

[2] A. Laarhoven. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing, Boston, MA, 1987.

[3] F. Hao and E. Zegura. On Scalable QoS Routing: Performance Evaluation of Topology Aggregation. In *IEEE INFOCOM*, pages 147–156, Mar. 2000.

[4] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Int'l J. Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[5] G. L. Bilbro and W. E. Snyder. Optimization of Function with Many Minima. *IEEE Trans. on Systems, Man, and Cybernetics*, 21(4):840–849, Aug. 1991.

[6] H. Shan, L. Oliker, and R. Biswas. Job SuperScheduler Architecture and Performance in Computational Grid Environments. In *ACM Conference on Supercomputing*, Nov. 2003.

[7] K. Hwang and Z. Xu. *Scalable Parallel Computing: Technology, Architecture, Programming*. McGraw Hill, New York, NY, 1998.

[8] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Int'l J. Supercomputer Applications*, 15(3):200–222, 2001.

[9] J. Moy. OSPF Version 2 . RFC 2328, Apr. 1998.

[10] J. S. Chase, D. C. Anderson, P. N. Thakar, A. Vahdat, and R. P. Doyle. Managing Energy and Server Resources in Hosting Centres. In *Symposium on Operating Systems Principles*, pages 103–116, Oct. 2001.

[11] K. Krauter, R. Buyya, and M. Maheswaran. A Taxonomy and Survey of Grid Resource Management Systems. *Software Practice and Experiance*, 32(2):135–164, Feb. 2002.

[12] L. Ingber. Simulated Annealing: Practice Versus Theory. *J. Mathematical Computing and Modelling*, 18(11):29–57, Dec. 1993.

[13] A. Oram. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly and Associates, Sebastopol, CA, 2001.

[14] P. Jogalekar and M. Woodside. Evaluating the Scalability of Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems*, 11(6):589–603, Mar. 2000.

[15] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, B. Park, and H. Song. Parsec: A Parallel Simulation Environment for Complex Systems. *IEEE Computer*, 31(10):77–85, Oct. 1998.

[16] R. Govindan and H. Tangmunarunkit. Heuristics for Internet Map Discovery. In *IEEE INFOCOM*, pages 1371–1380, Mar. 2000.

[17] S. Zhou. A Trace-Driven Simulation Study of Dynamic Load Balancing. *IEEE Trans. on Software Engineering*, 14(9):1327–1341, Sep. 1988.

[18] X. H. Sun and L. M. Ni. Scalable Problems and Memory-Bounded Speedup. *J. Parallel and Distributed Computing*, 19(1):27–37, Sep. 1993.

[19] T. Schroeder, S. Goddard, and B. Ramamurthy. Scalable Web Server Clustering Technologies. In *IEEE Network*, pages 38–45, Jun. 2000.

[20] V. P. Kumar and A. Gupta. Analyzing Scalability of Parallel Algorithms and Architectures. *J. of Parallel and Distributed Computing*, 22(3):379–391, 1994.

[21] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. M. Nahum. Locality-Aware Request Distribution in Cluster-based Network Servers. In *ACM Conference on Architectural Support for Programming Languages and Operating Systems*, pages 205–216, Oct. 1998.

[22] W. Cirne and F. Berman. A Comprehensive Model of the Supercomputer Workload. In *IEEE 4th Annual Workshop on Workload Characterization*, May 2001.

[23] W. Cirne and F. Berman. A Model for Moldable Supercomputer Jobs. In *International Parallel and Distributed Processing Symposium*, Apr. 2001.

[24] W. E. Leland and T. J. Ott. Load-balancing Heuristics and Process Behavior. In *ACM SIGMETRICS*, pages 54–69, May 1986.

Biographies

Arindam Mitra is pursuing a PhD degree from the University of Manitoba, Winnipeg, Manitoba, Canada. In 2002, he received a M.Sc. degree in computer science also from the same university. His research interests include computer networking, distributed computing, resource management systems for computing utilities, and peer-to-peer and heterogenous computing.

Muthucumaru Maheswaran is an Assistant Professor in the School of Computer Science at the McGill University, Montreal, Canada. In 1990, he received a BSc degree in electrical and electronic engineering from the University of Peradeniya, Sri Lanka. He received an MSEE degree in 1994 and a PhD in 1998, both from the school the School of the Electrical and Computer Engineering at Purdue University. He held a Fulbright scholarship during his tenure as an MSEE student at Purdue University. His research interests include computer architecture, distributed computing, heterogeneous computing, Internet and world wide web systems, metacomputing, mobile programs, network computing, parallel computing, resource management systems for metacomputing, and scientific computing. He is a member of the Eta Kappa Nu honorary society.

Shoukat Ali is an Assistant Professor in the Electrical and Computer Engineering Department at the University of Missouri-Rolla. In 1996, he received a B.S. degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan. He received an M.S. degree in 1999 and a PhD in 2003, both from School of the Electrical and Computer Engineering at Purdue University. 2002. His research interests include heterogeneous parallel and distributed computing and communication systems. He was the publicity cochair of the 11th IEEE Heterogeneous Computing Workshop (2002), and is on the program committee for the 12th IEEE Heterogeneous Computing Workshop (2003). He is a member of the IEEE, IEEE Computer Society, and ACM.

Variables	Values	Description
T_{CPU}	700 time units (constant)	Jobs with execution time $\leq T_{CPU}$ are LOCAL jobs. Jobs with execution time $> T_{CPU}$ are REMOTE jobs.
T_l	0.5 (constant)	Measurement for *threshold load* at a scheduler.
$U_b(jobid)$	variable	User benefit function. Measured as $k \times job\ run\ time$, where $k = [2, 5]$

Table 1. List of common variables and corresponding values used for all experiments.

Case 1: Scaling the RP by network size	
Scaling variables	- Network size in terms of number of nodes = sizeof[RMS] + sizeof[RP] - Workload (number of jobs arriving per unit time)
Scaling enablers	- Status update interval - Neighborhood set size - Network link delay

Table 2. Scaling by number of nodes. RMS increases proportionately with RP.

Case 2: Scaling the RP by resource service rate	
Scaling variables	- Resource service rate (number of jobs executed per unit time) - Workload (number of jobs arriving per unit time)
Scaling enablers	- Status update interval - Neighborhood set size - Network link delay

Table 3. Scaling by service rates.

Case 3: Scaling the RMS by number of status estimators	
Scaling variables	- Number of Status Estimators - Workload (number of jobs arriving per unit time)
Scaling enablers	- Status update interval - Neighborhood set size - Network link delay

Table 4. Scaling by number of estimators in the RMS. Estimators receive updates from RP.

Case 4: Scaling the RMS by L_p	
Scaling variables	- L_p: Number of neighbor schedulers being contacted for load balancing - Workload (number of jobs arriving per unit time)
Scaling enablers	- Status update interval - Interval for resource volunteering - Network link delay

Table 5. Scaling by number schedulers being probed or polled during scheduling.