[Kleinrock78] L.Kleinrock, *On Flow Control in Computer Networks*, ICC 1978, pp 27.2.1-27.2.5.

[Kleinrock79] L.Kleinrock, *Power and Deterministic Rules of Thumb for Probabilistic Problems in Computer Communications*,ICC 1979, pp 43.1.1-43.1.10.

[Rolia95] J.A.Rolia and K.C.Sevcik, *The Method of Layers*, IEEE Transactions on Software Engineering, Vol. 21, No. 8, August 1995, pp 689-700.

[Sarukkai95] S.R.Sarukkai, P.Mehra and R.J.Block, *Automated Scalability Analysis of Message-Passing Parallel Programs*, IEEE Parallel and Distributed Technology, Winter 1995, pp 21-32.

[Sun93] X.H.Sun and L.M.Ni, *Scalable Problems and Memory-Bounded Speedup*, Journal of Parallel and Distributed Computing, vol. 19, 1993, pp 27-37.

[Sun95] X.H.Sun and J.Zhu, *Performance Considerations of Shared Virtual Memory Machines*, IEEE Transactions on Parallel and Distributed Systems, Vol. 6, No. 11, November 1995, pp 1185-1194.

[Woodside95] C.M.Woodside, J.E.Neilson, D.C.Petriu and S.Majumdar, *The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software*, IEEE Transactions on Computers, Vol. 44, No. 1, January 1995, pp 20-34.

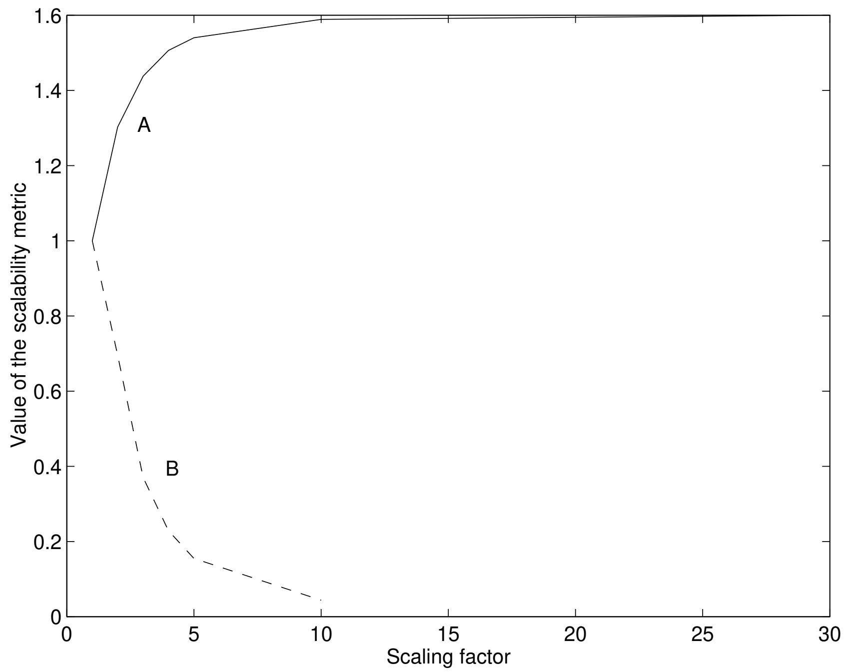


Fig.4. Evaluating the scale-up strategies.

which is based on the power metric defined by [Giessler78], is quite different from well-known scalability metrics such as Amdahl's law for parallel software. It captures the effect on both, the throughput capacity and the delay, due to a wide variety of changes which may be made to the system. The scalability measure is a function of many decisions in the configuration, such as allocation of tasks to nodes.

The next step is to analyze the scalability function only for a subset of "good" configurations, out of all the possible system configurations, obtained by optimizing with respect to some of the attributes. This is the subject of present research.

7. References

[Giessler78] A.Giessler, J.Hanle, A.Konig, E.Pade, *Free Buffer Allocation - An Investigation by Simulation*, Computer Networks, 1978, pp 191-208.

[Grama93] A.Y.Grama, A.Gupta and V.Kumar, *Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures*, IEEE Parallel and Distributed Technology, August 1993, pp 12-21.

Table 1: Mean Execution Times of the Entries (Remote/Local Procedures)

Entry (procedure entry point) offered by a task	No. of threads of the task	Execution time (sec)
User (clients)	10	Think time- 60
PA - E1	10	0.4
PA - E2	10	0.3
NRM - E1	1	1.0
CM - E1	5	1.0
RM - E1	10	0.9
RM - E2	10	0.8
DB - E1	4	0.5
DB - E2	4	0.5
DB - E3	4	0.5
SM - E1	5	2.0
SM - E2	5	1.5

Table 2: Relative Call Frequencies Between Entries

Remote / local procedure call	Relative Frequency
User --> PA-E1	1.0
User --> PA-E2	10.0
PA-E1 --> NRM-E1	1.0
PA-E2 --> CM-E1	1.0
NRM-E1 --> RM-E1	1.0
NRM-E1 --> DB-E2	4.0
NRM-E1 --> SM-E1	1.0
CM-E1 --> RM-E2	1.0
CM-E1 --> DB-E3	3.0
CM-E1 --> SM-E2	1.0
RM-E1 --> DB-E1	2.0
RM-E2 --> DB-E1	2.0

Scaling strategy B:

The strategy is the same as A,except that we do not scaleup the speed of the processor on which the database is executing. Fig. 4 (curve B) shows that $\psi < 1$, which means that the system cannot be scaled using this strategy, as the CPU on which the database manager executes soon becomes the bottleneck. Adding faster hardware elsewhere is not cost-effective although it results in marginal increase in the throughput for $R > 1$. Thus, we observe that while strategy A can scale up the system by a factor of 30, strategy B fails even below a factor of 2.

6. Conclusions:

A metric for the scalability of a distributed system from one configuration to another, is described. We say that a system is scalable from configuration S to S' if $\psi(S,S') > 1$ or $\psi \approx 1$. The metric,

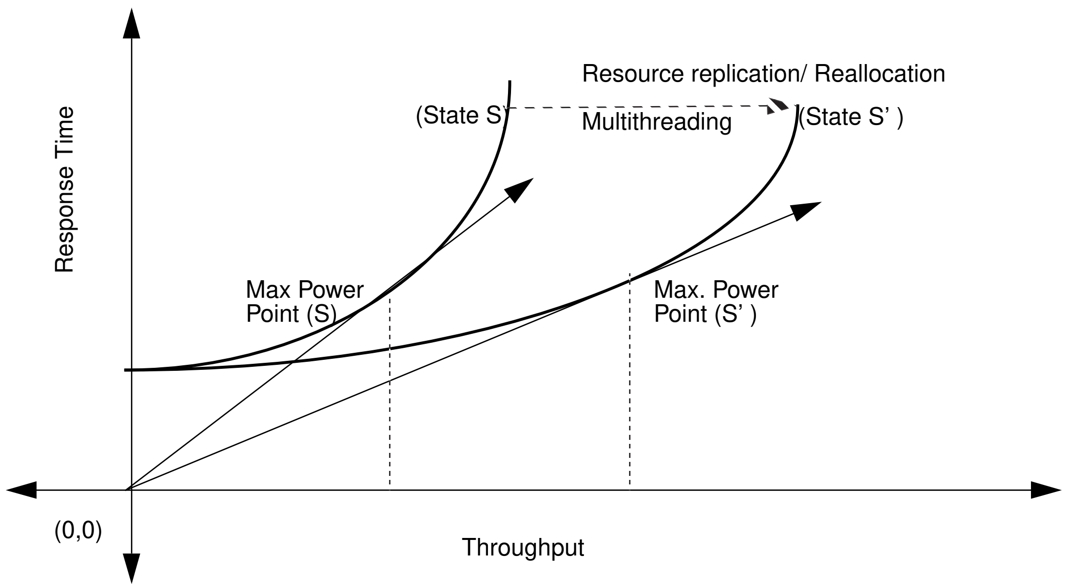


Fig.3 : Effects of change of state on maximum available power

The effectiveness of two scaling strategies is demonstrated here with respect to the example connection management system discussed in section 2. The layered queueing network parameters are as shown in the tables 1 and 2.

Scaling strategy A:

Assume a linear cost function for the hardware i.e., the hardware cost is directly proportional to the CPU speed. The ratio of the speeds of CPU1: CPU2: CPU3 (as shown in fig.2(a)) is 2:3:7 in the basic model. In order to support an increase in the number of active clients by a factor R, we replace all the CPUs by other CPUs which are R times as fast. With respect to the initial state described in the tables (1) and (2), the metric ψ shows that the system is scalable to any of the states reachable by the scaling strategy, as shown in fig. 4. The system shows a sharp scaleup ($\psi >$ 1) before a degree of software bottlenecking comes into the picture (at R = 5, approx.) and shows satisfactory scaleup even beyond that point.

curve. This fact captures important qualitative characteristics of the delay with respect to the throughput. Consider an open distributed system with power P = $\left(\frac{\lambda}{T(\lambda)}\right)$. Maximizing the power with respect to λ yields

$$\frac{\partial P}{\partial \lambda} = \frac{T - \left(\lambda \cdot \frac{\partial T}{\partial \lambda}\right)}{(T)^2} = 0 \Rightarrow \frac{T}{\lambda} = \frac{\partial T}{\partial \lambda} \qquad ....(6)$$

For a closed system, by maximizing the power function,

$$\frac{\partial \lambda}{\partial N} = \frac{\lambda}{2.N} \qquad ....(7)$$ at the maximum power point.

The flatter the curve, the better the system behaves when faced by an abrupt increase in the traffic. Thus, higher the maximum attainable power, the more robust the system configuration is. The cost function in the denominator in the definition of the scalability function makes sure that the investment to achieve this qualitative advantage in terms of the improved curve shape is justified. Fig. 3 illustrates this point.

5. Example of the use of the metric

The state variables that describe the scaling up of a distributed system may include:

software variables: collocation / distribution of the building blocks (software tasks), threading or creating multiple copies of the bottleneck servers, tuning the software for better efficiency, priority scheduling

middleware variables: thread priorities and other thread management functions, stacksizes for RPCs, trading algorithm, name/directory service algorithm, security features

hardware variables: adding extra or faster CPUs, disks, caches, communication links

Not all of these state variables describe adding extra resources, some merely describe optimizing the usage of existing resources (e.g. reallocation of software building blocks to the CPUs). Multithreading the servers can also be a cheap way to achieve extra power in many cases.

For a distributed (homogeneous or heterogeneous) system the scalability function from a state S to a new state S' is:

$$\psi(S, S') = \left(\left(\frac{\lambda(S')}{T(S')}\right) \cdot \frac{1}{C(S')}\right) \div \left(\left(\frac{\lambda(S)}{T(S)}\right) \cdot \frac{1}{C(S)}\right) \qquad ....(4)$$

Where: $\left(\frac{\lambda(S')}{T(S')}\right)$ = the power associated with the state S' (and similarly for S),

C(S'), C(S) = the cost of achieving the states S' and S respectively.

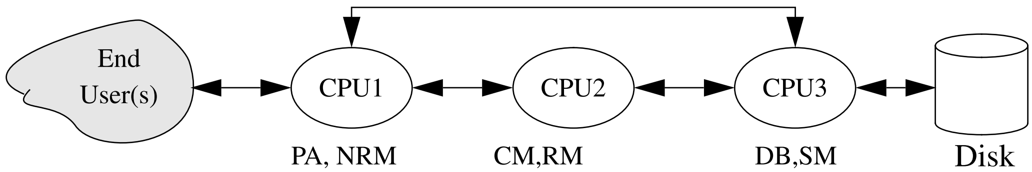
The cost function could be expressed in the units of the total number of available flops, the total amount of investment in dollars associated with the system configuration, weighted sum of the number of hardware and software modules, total RAM available in a configuration, etc. The precise choice of the cost function will however be outside the scope of this paper. In addition to this definition, an optional upper bound for the delay could be specified for a state to be acceptable. Overall, the scalability metric measures the return on investment, for increasing the traffic handling capacity of the system, or improved QoS, or both.

For closed systems, applying Little's result ($N = \lambda . T$) gives:

$$\psi(S, S') = \left(\frac{(\lambda(S'))^2}{N(S')} \cdot \frac{1}{C(S')}\right) \div \left(\frac{(\lambda(S))^2}{N(S)} \cdot \frac{1}{C(S)}\right) \qquad ....(5)$$

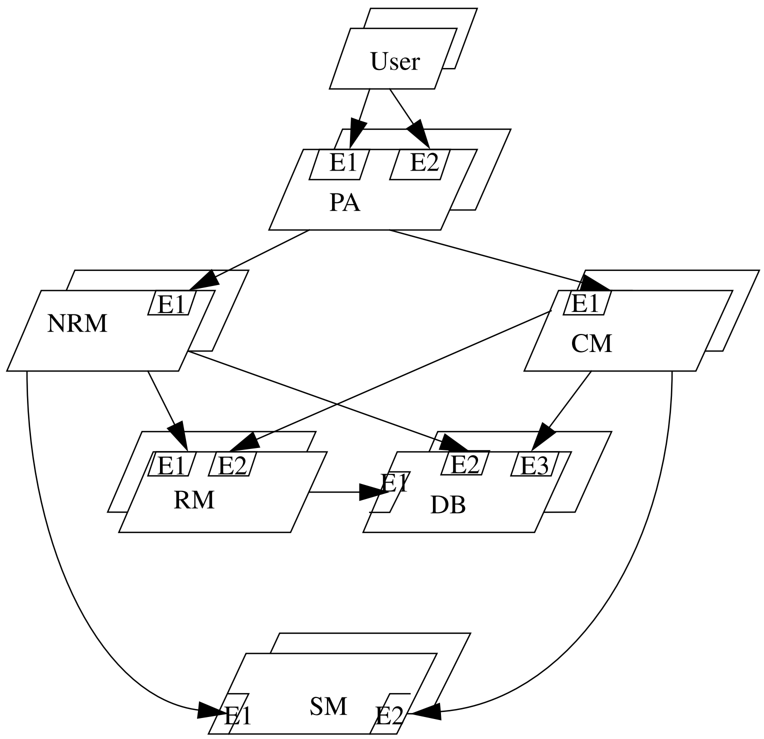
Suppose that a change in the system configuration increases the throughput, but causes too large mean delays, the scale up may not be efficient. Similarly, for bounded delays, the possible throughput levels might be unacceptably low. Such effects of scaling up a system are captured by the above definition of scalability between any two system states.

The most striking characteristic of the power metric, as discussed in [Kleinrock78], is that for *any* delay function T (λ), the maximum power point is the place where the tangent from the origin touches the T-λ curve. The maximum attainable power for a configuration is the inverse of the slope of this tangent. That is, the maximum power operating point is at the "knee" of this



A workload model view of the queueing network corresponding to the connection management system

Fig. 2(a)



Layered queueing network model of the connection management system

Fig.2(b) Queueing models of the example connection management system

The *state* of a distributed system is a set of attributes which define the configuration of its resources. An attribute could describe application or middleware, the hardware, or the traffic. Effects of changing any of the attributes are captured by the power metric. The power function for a queueing network rewards concurrency and penalizes communication and other overheads. *If the power per invested dollar can be improved (or maintained constant) by evolving the system configuration, the distributed system is said to be scalable from its old state to the new state*.

- The system resources are shared for processing the concurrent requests from different clients, leading to queueing at the software and the hardware resources.

- There are more variables affecting the scalability of the system. e.g. heterogeneity, non-regular communication topology, non-regular software architectures, complex communication protocols, etc. Thus, there could be several potential sources of performance limits.

For these reasons, the metric (1) is unsuitable and so are the other metrics mentioned.

4. A Scalability Metric for Distributed Software Systems:

A new scalability metric will be defined relative to the system's response times and throughputs, which in turn may be evaluated by a queueing model or a layered queueing model of the system. Fig. 2(a) describes a queueing model for the example, with a certain allocation of objects to processors. Fig. 2(b) describes the layered model. The layered model has more information in it. It identifies the services offered by one software module to another, and the response times for them. It also determines the contention that may occur at a software server, as well as at the hardware, and the effects of software resources such as a limited pool of concurrently executing threads. The model, as described in [Woodside95],[Rolia95] will be used to evaluate the performance of this example in different configurations.

The objective of scaling up a distributed system is either to support more throughput for a fixed response time, to reduce the response time for a given throughput, or a combination of these. If none of these can be achieved by reconfiguring the system and/or adding more resources, we say that it cannot be scaled. The combined effect of throughput and response time can be evaluated as the "*power*" metric, defined as[Giessler78]:

$$P = \frac{Throughput(\lambda)}{ResponseTime(T)} \quad ....(3)$$

This metric has been used by Kleinrock in [Kleinrock78], [Kleinrock79] for deriving deterministic rules of thumb for probabilistic problems in computer communications. The power function has the attractive property of increasing proportional to the throughput for fixed response time, and inversely with the response time, for a fixed throughput.

tion time) - the processing demands of the given application are held constant (independent of the number of processors) while evaluating the abovementioned ratio, called Amdahl's law [Grama93], [Sun93].

$$\text{Fixed size speedup} = \frac{W \cdot C_s}{\sum_{i=1}^{p} \frac{W_i}{i} \cdot C_p} \quad \text{................(1)}$$

Where: p = number of parallel processors,

W_i = the total work done when exactly i processors are active,

W = the total work to be done = $\sum_i W_i$

C_s, C_p = sequential and parallel execution times per operation step, respectively,

Another possibility is to analyze the *fixed time speedup* (time constrained scaling). In this case, as the machine parameters (e.g. the number of processors, cache size for each processor, bus bandwidth, etc.) are scaled, the processing demands of the application are correspondingly scaled such that the application execution time remains fixed. Such a scalability analysis assumes that the user is interested in some fixed range of the execution times, below which there is no need/use of extra investment and above which, the delays are unacceptable. The fixed-time scaled speedup is called Gustafson's law [Grama93], [Sun93].

$$\text{Fixed time speedup} = \frac{\text{(Sum of the work done by all the parallel processors)}}{\text{(Work done by a sequential system in the same time)}} \quad \text{.........(2)}$$

For the definitions and analytical expressions for isospeed, isoefficiency scalability, etc., we refer the readers to the references mentioned in section 1. This analysis cannot be extended to the distributed systems directly, because the distributed systems are different from the multiprocessor-based parallel systems in the following ways:

- The distributed systems typically handle many concurrent requests for service, instead of running one instance to completion. In addition, there may be concurrency within a thread of execution of any particular client.

and administering multiple copies of the connection manager or the database be justified against the enhancement in the QoS experienced by the clients? In short, does the system produce a satisfactory return on any extra investment in augmenting any of the modules in the client-server hierarchy? We will define a metric which evaluates the gain due to the scale-up versus the investment incurred in doing so.

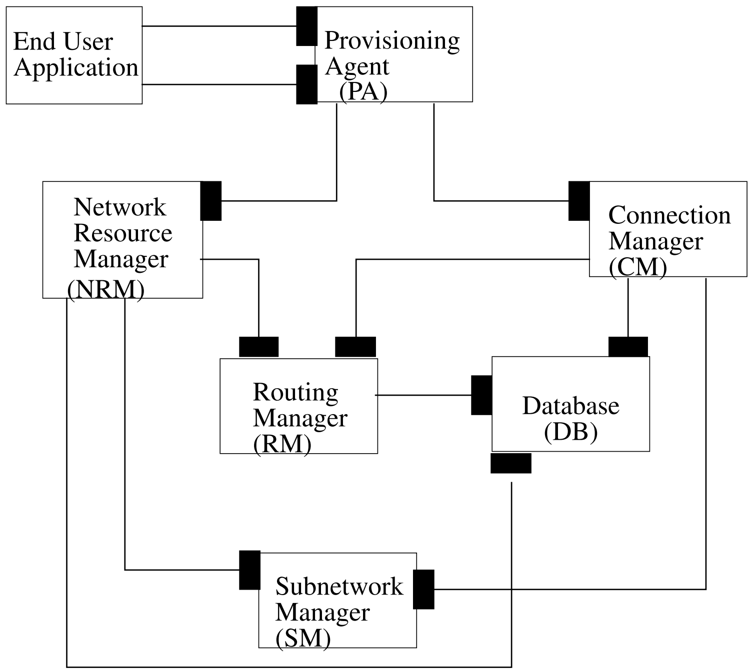


Fig.1 An Example Connection Management System

3. Scalability Analysis for Multiprocessor Systems

Scalability issues have been extensively studied for parallel software executing on bus-based or grid-connected multiprocessor systems. In this section we summarize some of the scalability ideas found useful in the multiprocessor domain, and show that they cannot be directly extended to the distributed domain, thus necessitating a new metric described in the next section.

Parallel software typically runs to completion on symmetric multiprocessors with a regular interconnection structure, and the software structure mirrors the hardware. The solution time for a given problem is related to the problem size and the number of processors. While analyzing *fixed size speedup*, (defined as the ratio of sequential execution time to the parallel execu-

reads the allocation state of the network from the database server. If the best possible path is unavailable due to previous allocation or due to the QoS requirement, it chooses the next best available path. The path may span several switches, links, and access groups, which are managed by the Subnetwork Manager object described below.

Note that if the route spans the network owned by different entities, negotiations have to be carried out by the network resource manager, failing which, the routing function has to be invoked again.

The database (DB): It maintains the global state of all the network resources, their allocation map, connections, and the information required for the routing function. With any change in the system state, the database is updated atomically.

To scale the system up, the database could be replicated and distributed. We assume that database access conflicts, race conditions etc. are not a major performance concern in the system under consideration, although with more effort, these could be modelled.

The Subnetwork Manager (SM): The subnetwork consists of switches attached to the links through access groups. The subnetwork manager establishes and releases the connections between specified endpoints. During the resource allocation/deallocation phase, the SM is invoked by the NRM and during the connection establishment/release, the SM is invoked by the connection manager. Any change in the connection map is synchronized with the database state by the object that calls the SM.

In its simplest form, the SM might manage just a single switch. (The connection establishment corresponds to designating one port as the input and another as the output in the connection table) In more complex forms, the subnetwork could be a series of trunks/fibres and switches.

Scalability Questions:

Can adding a fast machine to the network buy a proportional degree of performance of the above system (after suitably reallocating the objects)? Can the extra investment in maintaining

vices offered by the provisioning agent to the end user application include reserving the bandwidth in advance for multipoint session(s), setting up and tearing down the connections between two or more of these points, and freeing the reserved resources.

When a request for bandwidth allocation by an end user application is successfully processed by a provisioning-agent, a set of identifiers for each access-point is returned which could potentially be a connection end point (e.g. in the case of an underlying ATM network, a UNI-VPI combination is returned). Using these identifiers, the End Users can set up and release connections with other End User sites.

<u>Provisioning Agent: (PA)</u>: This object provides a secure interface to the End User. It carries out the authentication and authorization checks on client requests, and processes and forwards the user RPCs to the network resource manager for reserving/releasing bandwidth and to the connection manager for setting up or releasing connections between designated end user sites.

<u>Network Resource Manager (NRM)</u>: For provisioning the required bandwidth and the required quality of service between the user sites, it invokes the routing manager and chooses the network path. Note that this function, in general, can be quite complex if the ownership of the underlying network is partitioned among many companies. In such a case, peer-to-peer communication between Network Resource Managers corresponding to each network partition is required for service and quality of service negotiation.

If a feasible path is available, the resource manager initiates a transaction that recursively invokes the subnetwork manager for setting up the connections in the underlying subnetwork (which could be a single switch in the simplest case). When the transaction commits, the NRM updates the database accordingly.

During the freeing of the network bandwidth, it follows an exactly reverse path as above.

<u>Routing Manager (RM)</u>: It implements the routing algorithms for the required connections such that some cost function (e.g. the number of hops between the two endpoints) is minimized. RM

support a larger number of clients.

Section 2 of this paper describes an example of a TINA-compliant connection management system and the associated scalability concerns. Section 3 presents an overview of the existing scalability metrics for parallel software systems, and describes the limitations when applied to distributed software. Section 4 defines and analyzes the new metric for scalability analysis of distributed software systems. Section 5 illustrates the usefulness of this metric, while making a decision to gradually evolve the system. Section 6 concludes this paper and lists our long-term research objectives in this area.

2. An Example Connection Management System

Fig.1 shows the computational objects and their interactions in a TINA-compliant connection management system which executes on the Distributed Processing Environment (DPE). The DPE provides distribution transparency to the computational objects. The interfaces shown in the diagram are invoked using remote procedure calls. Some underlying objects like the management software for the network elements, communication medium and interaction between its components, etc. have been abstracted out in order to focus on the software aspects of scalability.

The computational objects include the End User Application, Provisioning Agent, Resource Manager, Routing Function, Subnetwork Connection Manager and the Database Management system, which are briefly described in this section. This distributed system is used to implement the two phases of connection management described below:

Phase 1: Allocation/Deallocation of bandwidth and getting appropriate end-point identifiers for setting up future connections.

Phase 2: Setting up/tearing down the connections on previously allocated resources.

<u>The End User Application</u>: The End User object in Fig.1 has a generic role which may be filled by many user objects. The End User object uses predefined codes (access groups that are like telephone numbers) for communicating with the provisioning agent, to achieve phases 1 & 2. The ser-

of operation in proportion to their cost, the system will become uneconomical to operate. This is the sense of scalability we wish to address.

While scalability has been studied for massively parallel systems, the issues are somewhat different for a distributed system. This work defines a metric for the scalability of distributed systems, based on performance and cost. Using an example for connection management, the usefulness and the intuitive basis for the metric are demonstrated. The proposed metric has a close relationship with the laws of Amdahl and Gustafson for parallel software scalability.

For parallel computation, several different metrics have been defined. They can be used to describe the effectiveness of changing various parameters of the system (e.g. [Sun95], [Grama93], [Sun93], [Sarukkai95] etc.), with respect to given algorithm-machine combinations. Such analysis may be useful (e.g. [Sarukkai95]) for selecting the most scalable algorithm for a given paradigm.

However, distributed systems are usually more complex than the parallel systems. They typically have heterogeneous processors and interconnection mechanisms, an irregular topology of connections (compared to homogeneous processors and communications and a regular topology of connections in most parallel systems), and they handle multiple concurrent inputs. Geographic separation and heterogeneity introduce overhead operations for protocols, security and trading, which change sharply under changes in interconnection and distribution. Merely increasing the hardware resources may not scale up the system's performance if the bottleneck still lies in the software/middleware. Scaling up a software resource affects the overall workload patterns in its clients as well as servers.

This paper defines a scalability metric that relates the *power* of a system to the cost of obtaining that power. It is easy to evaluate and use. This metric could help making decisions about evolving the system along the most economical path. It could also be used for tuning the system for a given set of resources, in order to maximize the concurrency between its hardware and software modules while minimizing the communication and other overhead costs, thus enabling it to

A Scalability Metric for Distributed Computing Applications in Telecommunications

P.P.Jogalekar, C.M.Woodside
Department of Systems and Computer Engineering,
Carleton University, Ottawa, Canada.
tel: 1-613-788-5721, fax: 1-613-788-5727
email: ppj@sce.carleton.ca , cmw@sce.carleton.ca

Abstract

Telecommunications systems must be scalable, meaning that they can be economically deployed in large and small versions. The configurations will differ in their hardware, software, interconnection and traffic parameters. Many future systems will be implemented by distributed software. A new metric is defined to evaluate the scalability of a distributed system from one configuration to another, and it is demonstrated on a connection management example based on the TINA Distributed Processing Environment. The new metric is quite different from metrics for massively parallel systems. It accommodates complex patterns of change and successfully captures intuitive notions of scaleup in a simple formula.

1. Introduction

Distributed software is a key component of modern telecommunication systems. Distributed software applications may include telecommunications services management, customer network management, connection management, fault management, etc. These systems are composed of several clients and servers (or “building blocks”) which communicate with each other over a distributed computing environment, typically using remote procedure calls.

This software must be scalable, that is, it must be capable of being deployed efficiently at both large and small scales. Over time it must also be possible to add capacity to either support more users, or enhance the quality of service, or both. If added resources do not increase the scale

[Kleinrock78] L.Kleinrock, *On Flow Control in Computer Networks*, ICC 1978, pp 27.2.1-27.2.5.

[Kleinrock79] L.Kleinrock, *Power and Deterministic Rules of Thumb for Probabilistic Problems in Computer Communications*,ICC 1979, pp 43.1.1-43.1.10.

[Rolia95] J.A.Rolia and K.C.Sevcik, *The Method of Layers*, IEEE Transactions on Software Engineering, Vol. 21, No. 8, August 1995, pp 689-700.

[Sarukkai95] S.R.Sarukkai, P.Mehra and R.J.Block, *Automated Scalability Analysis of Message-Passing Parallel Programs*, IEEE Parallel and Distributed Technology, Winter 1995, pp 21-32.

[Sun93] X.H.Sun and L.M.Ni, *Scalable Problems and Memory-Bounded Speedup*, Journal of Parallel and Distributed Computing, vol. 19, 1993, pp 27-37.

[Sun95] X.H.Sun and J.Zhu, *Performance Considerations of Shared Virtual Memory Machines*, IEEE Transactions on Parallel and Distributed Systems, Vol. 6, No. 11, November 1995, pp 1185-1194.

[Woodside95] C.M.Woodside, J.E.Neilson, D.C.Petriu and S.Majumdar, *The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software*, IEEE Transactions on Computers, Vol. 44, No. 1, January 1995, pp 20-34.

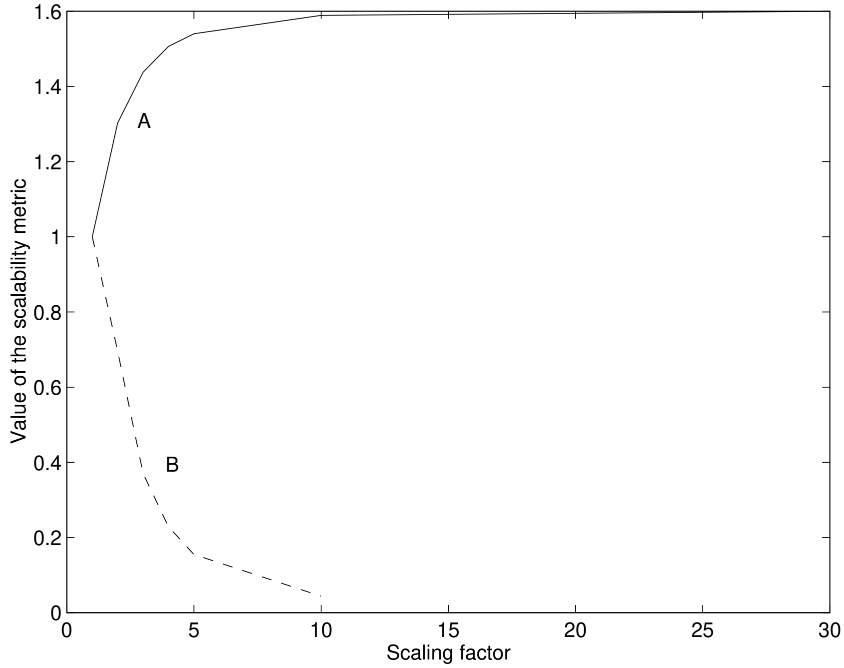


Fig.4. Evaluating the scale-up strategies.

which is based on the power metric defined by [Giessler78], is quite different from well-known scalability metrics such as Amdahl's law for parallel software. It captures the effect on both, the throughput capacity and the delay, due to a wide variety of changes which may be made to the system. The scalability measure is a function of many decisions in the configuration, such as allocation of tasks to nodes.

The next step is to analyze the scalability function only for a subset of "good" configurations, out of all the possible system configurations, obtained by optimizing with respect to some of the attributes. This is the subject of present research.

7. References

[Giessler78] A.Giessler, J.Hanle, A.Konig, E.Pade, *Free Buffer Allocation - An Investigation by Simulation*, Computer Networks, 1978, pp 191-208.

[Grama93] A.Y.Grama, A.Gupta and V.Kumar, *Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures*, IEEE Parallel and Distributed Technology, August 1993, pp 12-21.

Table 1: Mean Execution Times of the Entries (Remote/Local Procedures)

Entry (procedure entry point) offered by a task	No. of threads of the task	Execution time (sec)
User (clients)	10	Think time- 60
PA - E1	10	0.4
PA - E2	10	0.3
NRM - E1	1	1.0
CM - E1	5	1.0
RM - E1	10	0.9
RM - E2	10	0.8
DB - E1	4	0.5
DB - E2	4	0.5
DB - E3	4	0.5
SM - E1	5	2.0
SM - E2	5	1.5

Table 2: Relative Call Frequencies Between Entries

Remote / local procedure call	Relative Frequency
User --> PA-E1	1.0
User --> PA-E2	10.0
PA-E1 --> NRM-E1	1.0
PA-E2 --> CM-E1	1.0
NRM-E1 --> RM-E1	1.0
NRM-E1 --> DB-E2	4.0
NRM-E1 --> SM-E1	1.0
CM-E1 --> RM-E2	1.0
CM-E1 --> DB-E3	3.0
CM-E1 --> SM-E2	1.0
RM-E1 --> DB-E1	2.0
RM-E2 --> DB-E1	2.0

Scaling strategy B:

The strategy is the same as A,except that we do not scaleup the speed of the processor on which the database is executing. Fig. 4 (curve B) shows that $\psi < 1$, which means that the system cannot be scaled using this strategy, as the CPU on which the database manager executes soon becomes the bottleneck. Adding faster hardware elsewhere is not cost-effective although it results in marginal increase in the throughput for $R > 1$. Thus, we observe that while strategy A can scale up the system by a factor of 30, strategy B fails even below a factor of 2.

6. Conclusions:

A metric for the scalability of a distributed system from one configuration to another, is described. We say that a system is scalable from configuration S to S’ if $\psi(S,S') > 1$ or $\psi \approx 1$. The metric,

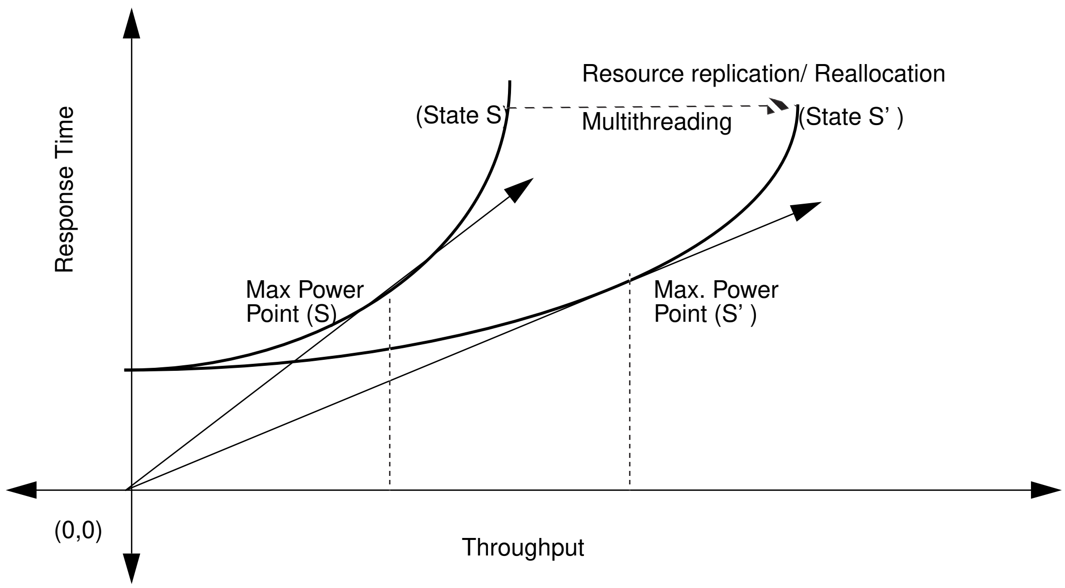


Fig.3 : Effects of change of state on maximum available power

The effectiveness of two scaling strategies is demonstrated here with respect to the example connection management system discussed in section 2. The layered queueing network parameters are as shown in the tables 1 and 2.

Scaling strategy A:

Assume a linear cost function for the hardware i.e., the hardware cost is directly proportional to the CPU speed. The ratio of the speeds of CPU1: CPU2: CPU3 (as shown in fig.2(a)) is 2:3:7 in the basic model. In order to support an increase in the number of active clients by a factor R, we replace all the CPUs by other CPUs which are R times as fast. With respect to the initial state described in the tables (1) and (2), the metric ψ shows that the system is scalable to any of the states reachable by the scaling strategy, as shown in fig. 4. The system shows a sharp scaleup ($\psi > 1$) before a degree of software bottlenecking comes into the picture (at R = 5, approx.) and shows satisfactory scaleup even beyond that point.

curve. This fact captures important qualitative characteristics of the delay with respect to the throughput. Consider an open distributed system with power P = $\left(\frac{\lambda}{T(\lambda)}\right)$. Maximizing the power with respect to λ yields

$$\frac{\partial P}{\partial \lambda} = \frac{T - \left(\lambda \cdot \frac{\partial T}{\partial \lambda}\right)}{(T)^2} = 0 \Rightarrow \frac{T}{\lambda} = \frac{\partial T}{\partial \lambda} \qquad ....(6)$$

For a closed system, by maximizing the power function,

$$\frac{\partial \lambda}{\partial N} = \frac{\lambda}{2.N} \qquad ....(7)$$ at the maximum power point.

The flatter the curve, the better the system behaves when faced by an abrupt increase in the traffic. Thus, higher the maximum attainable power, the more robust the system configuration is. The cost function in the denominator in the definition of the scalability function makes sure that the investment to achieve this qualitative advantage in terms of the improved curve shape is justified. Fig. 3 illustrates this point.

5. Example of the use of the metric

The state variables that describe the scaling up of a distributed system may include:

software variables: collocation / distribution of the building blocks (software tasks), threading or creating multiple copies of the bottleneck servers, tuning the software for better efficiency, priority scheduling

middleware variables: thread priorities and other thread management functions, stacksizes for RPCs, trading algorithm, name/directory service algorithm, security features

hardware variables: adding extra or faster CPUs, disks, caches, communication links

Not all of these state variables describe adding extra resources, some merely describe optimizing the usage of existing resources (e.g. reallocation of software building blocks to the CPUs). Multithreading the servers can also be a cheap way to achieve extra power in many cases.

For a distributed (homogeneous or heterogeneous) system the scalability function from a state S to a new state S' is:

$$\psi(S, S') = \left(\left(\frac{\lambda(S')}{T(S')}\right) \cdot \frac{1}{C(S')}\right) \div \left(\left(\frac{\lambda(S)}{T(S)}\right) \cdot \frac{1}{C(S)}\right) \qquad ....(4)$$

Where: $\left(\frac{\lambda(S')}{T(S')}\right)$ = the power associated with the state S' (and similarly for S),

C(S'), C(S) = the cost of achieving the states S' and S respectively.

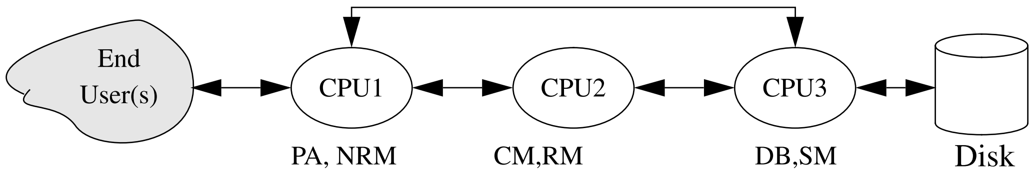
The cost function could be expressed in the units of the total number of available flops, the total amount of investment in dollars associated with the system configuration, weighted sum of the number of hardware and software modules, total RAM available in a configuration, etc. The precise choice of the cost function will however be outside the scope of this paper. In addition to this definition, an optional upper bound for the delay could be specified for a state to be acceptable. Overall, the scalability metric measures the return on investment, for increasing the traffic handling capacity of the system, or improved QoS, or both.

For closed systems, applying Little's result ($N = \lambda . T$) gives:

$$\psi(S, S') = \left(\frac{(\lambda(S'))^2}{N(S')} \cdot \frac{1}{C(S')}\right) \div \left(\frac{(\lambda(S))^2}{N(S)} \cdot \frac{1}{C(S)}\right) \qquad ....(5)$$

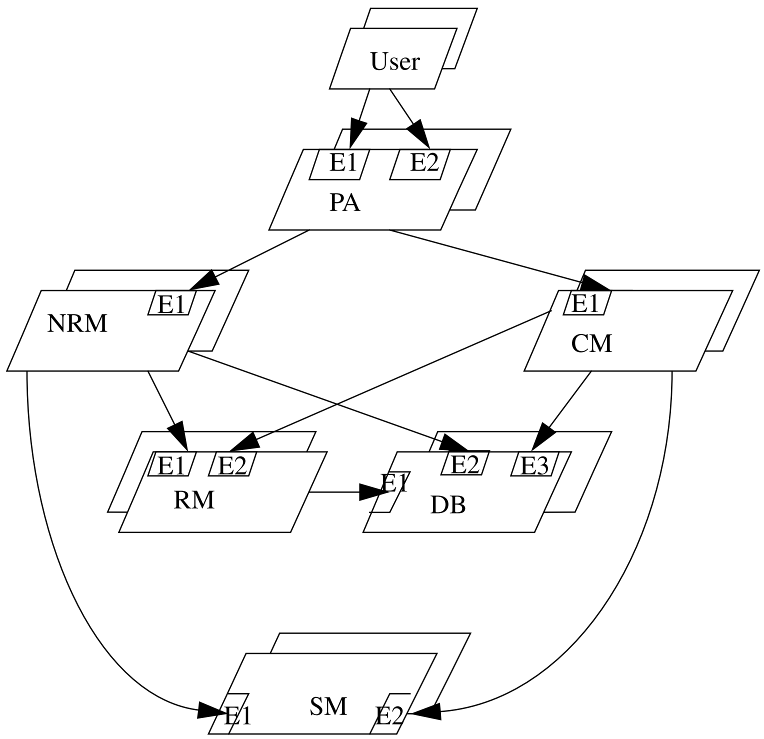
Suppose that a change in the system configuration increases the throughput, but causes too large mean delays, the scale up may not be efficient. Similarly, for bounded delays, the possible throughput levels might be unacceptably low. Such effects of scaling up a system are captured by the above definition of scalability between any two system states.

The most striking characteristic of the power metric, as discussed in [Kleinrock78], is that for *any* delay function T (λ), the maximum power point is the place where the tangent from the origin touches the T-λ curve. The maximum attainable power for a configuration is the inverse of the slope of this tangent. That is, the maximum power operating point is at the "knee" of this



A workload model view of the queueing network corresponding to the connection management system

Fig. 2(a)



Layered queueing network model of the connection management system

Fig.2(b) Queueing models of the example connection management system

The *state* of a distributed system is a set of attributes which define the configuration of its resources. An attribute could describe application or middleware, the hardware, or the traffic. Effects of changing any of the attributes are captured by the power metric. The power function for a queueing network rewards concurrency and penalizes communication and other overheads. *If the power per invested dollar can be improved (or maintained constant) by evolving the system configuration, the distributed system is said to be scalable from its old state to the new state*.

- The system resources are shared for processing the concurrent requests from different clients, leading to queueing at the software and the hardware resources.

- There are more variables affecting the scalability of the system. e.g. heterogeneity, non-regular communication topology, non-regular software architectures, complex communication protocols, etc. Thus, there could be several potential sources of performance limits.

For these reasons, the metric (1) is unsuitable and so are the other metrics mentioned.

4. A Scalability Metric for Distributed Software Systems:

A new scalability metric will be defined relative to the system's response times and throughputs, which in turn may be evaluated by a queueing model or a layered queueing model of the system. Fig. 2(a) describes a queueing model for the example, with a certain allocation of objects to processors. Fig. 2(b) describes the layered model. The layered model has more information in it. It identifies the services offered by one software module to another, and the response times for them. It also determines the contention that may occur at a software server, as well as at the hardware, and the effects of software resources such as a limited pool of concurrently executing threads. The model, as described in [Woodside95],[Rolia95] will be used to evaluate the performance of this example in different configurations.

The objective of scaling up a distributed system is either to support more throughput for a fixed response time, to reduce the response time for a given throughput, or a combination of these. If none of these can be achieved by reconfiguring the system and/or adding more resources, we say that it cannot be scaled. The combined effect of throughput and response time can be evaluated as the "*power*" metric, defined as[Giessler78]:

$$P = \frac{Throughput(\lambda)}{ResponseTime(T)} \qquad \text{.................................(3)}$$

This metric has been used by Kleinrock in [Kleinrock78], [Kleinrock79] for deriving deterministic rules of thumb for probabilistic problems in computer communications. The power function has the attractive property of increasing proportional to the throughput for fixed response time, and inversely with the response time, for a fixed throughput.

tion time) - the processing demands of the given application are held constant (independent of the number of processors) while evaluating the abovementioned ratio, called Amdahl's law [Grama93], [Sun93].

$$\text{Fixed size speedup} = \frac{W \cdot C_s}{\sum_{i=1}^{p} \frac{W_i}{i} \cdot C_p} \quad \text{................(1)}$$

Where: p = number of parallel processors,

W_i = the total work done when exactly i processors are active,

W = the total work to be done = $\sum_i W_i$

C_s, C_p = sequential and parallel execution times per operation step, respectively,

Another possibility is to analyze the *fixed time speedup* (time constrained scaling). In this case, as the machine parameters (e.g. the number of processors, cache size for each processor, bus bandwidth, etc.) are scaled, the processing demands of the application are correspondingly scaled such that the application execution time remains fixed. Such a scalability analysis assumes that the user is interested in some fixed range of the execution times, below which there is no need/use of extra investment and above which, the delays are unacceptable. The fixed-time scaled speedup is called Gustafson's law [Grama93], [Sun93].

$$\text{Fixed time speedup} = \frac{\text{(Sum of the work done by all the parallel processors)}}{\text{(Work done by a sequential system in the same time)}} \quad \text{.........(2)}$$

For the definitions and analytical expressions for isospeed, isoefficiency scalability, etc., we refer the readers to the references mentioned in section 1. This analysis cannot be extended to the distributed systems directly, because the distributed systems are different from the multiprocessor-based parallel systems in the following ways:

- The distributed systems typically handle many concurrent requests for service, instead of running one instance to completion. In addition, there may be concurrency within a thread of execution of any particular client.

and administering multiple copies of the connection manager or the database be justified against the enhancement in the QoS experienced by the clients? In short, does the system produce a satisfactory return on any extra investment in augmenting any of the modules in the client-server hierarchy? We will define a metric which evaluates the gain due to the scale-up versus the investment incurred in doing so.

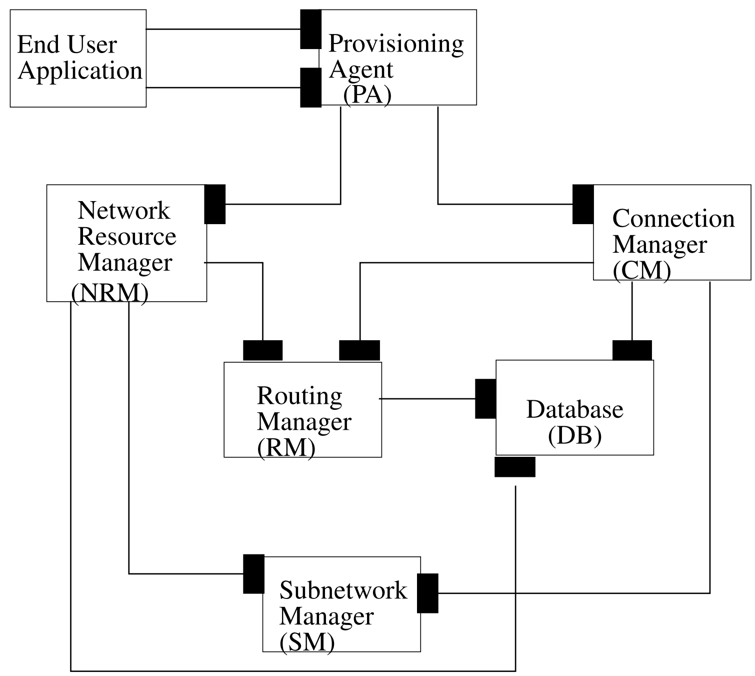


Fig.1 An Example Connection Management System

3. Scalability Analysis for Multiprocessor Systems

Scalability issues have been extensively studied for parallel software executing on bus-based or grid-connected multiprocessor systems. In this section we summarize some of the scalability ideas found useful in the multiprocessor domain, and show that they cannot be directly extended to the distributed domain, thus necessitating a new metric described in the next section.

Parallel software typically runs to completion on symmetric multiprocessors with a regular interconnection structure, and the software structure mirrors the hardware. The solution time for a given problem is related to the problem size and the number of processors. While analyzing *fixed size speedup*, (defined as the ratio of sequential execution time to the parallel execu-

reads the allocation state of the network from the database server. If the best possible path is unavailable due to previous allocation or due to the QoS requirement, it chooses the next best available path. The path may span several switches, links, and access groups, which are managed by the Subnetwork Manager object described below.

Note that if the route spans the network owned by different entities, negotiations have to be carried out by the network resource manager, failing which, the routing function has to be invoked again.

<u>The database (DB):</u> It maintains the global state of all the network resources, their allocation map, connections, and the information required for the routing function. With any change in the system state, the database is updated atomically.

To scale the system up, the database could be replicated and distributed. We assume that database access conflicts, race conditions etc. are not a major performance concern in the system under consideration, although with more effort, these could be modelled.

<u>The Subnetwork Manager (SM):</u> The subnetwork consists of switches attached to the links through access groups. The subnetwork manager establishes and releases the connections between specified endpoints. During the resource allocation/deallocation phase, the SM is invoked by the NRM and during the connection establishment/release, the SM is invoked by the connection manager. Any change in the connection map is synchronized with the database state by the object that calls the SM.

In its simplest form, the SM might manage just a single switch. (The connection establishment corresponds to designating one port as the input and another as the output in the connection table) In more complex forms, the subnetwork could be a series of trunks/fibres and switches.

Scalability Questions:

Can adding a fast machine to the network buy a proportional degree of performance of the above system (after suitably reallocating the objects)? Can the extra investment in maintaining

vices offered by the provisioning agent to the end user application include reserving the bandwidth in advance for multipoint session(s), setting up and tearing down the connections between two or more of these points, and freeing the reserved resources.

When a request for bandwidth allocation by an end user application is successfully processed by a provisioning-agent, a set of identifiers for each access-point is returned which could potentially be a connection end point (e.g. in the case of an underlying ATM network, a UNI-VPI combination is returned). Using these identifiers, the End Users can set up and release connections with other End User sites.

<u>Provisioning Agent: (PA)</u>: This object provides a secure interface to the End User. It carries out the authentication and authorization checks on client requests, and processes and forwards the user RPCs to the network resource manager for reserving/releasing bandwidth and to the connection manager for setting up or releasing connections between designated end user sites.

<u>Network Resource Manager (NRM)</u>: For provisioning the required bandwidth and the required quality of service between the user sites, it invokes the routing manager and chooses the network path. Note that this function, in general, can be quite complex if the ownership of the underlying network is partitioned among many companies. In such a case, peer-to-peer communication between Network Resource Managers corresponding to each network partition is required for service and quality of service negotiation.

If a feasible path is available, the resource manager initiates a transaction that recursively invokes the subnetwork manager for setting up the connections in the underlying subnetwork (which could be a single switch in the simplest case). When the transaction commits, the NRM updates the database accordingly.

During the freeing of the network bandwidth, it follows an exactly reverse path as above.

<u>Routing Manager (RM)</u>: It implements the routing algorithms for the required connections such that some cost function (e.g. the number of hops between the two endpoints) is minimized. RM

support a larger number of clients.

Section 2 of this paper describes an example of a TINA-compliant connection management system and the associated scalability concerns. Section 3 presents an overview of the existing scalability metrics for parallel software systems, and describes the limitations when applied to distributed software. Section 4 defines and analyzes the new metric for scalability analysis of distributed software systems. Section 5 illustrates the usefulness of this metric, while making a decision to gradually evolve the system. Section 6 concludes this paper and lists our long-term research objectives in this area.

2. An Example Connection Management System

Fig.1 shows the computational objects and their interactions in a TINA-compliant connection management system which executes on the Distributed Processing Environment (DPE). The DPE provides distribution transparency to the computational objects. The interfaces shown in the diagram are invoked using remote procedure calls. Some underlying objects like the management software for the network elements, communication medium and interaction between its components, etc. have been abstracted out in order to focus on the software aspects of scalability.

The computational objects include the End User Application, Provisioning Agent, Resource Manager, Routing Function, Subnetwork Connection Manager and the Database Management system, which are briefly described in this section. This distributed system is used to implement the two phases of connection management described below:

Phase 1: Allocation/Deallocation of bandwidth and getting appropriate end-point identifiers for setting up future connections.

Phase 2: Setting up/tearing down the connections on previously allocated resources.

<u>The End User Application</u>: The End User object in Fig.1 has a generic role which may be filled by many user objects. The End User object uses predefined codes (access groups that are like telephone numbers) for communicating with the provisioning agent, to achieve phases 1 & 2. The ser-

of operation in proportion to their cost, the system will become uneconomical to operate. This is the sense of scalability we wish to address.

While scalability has been studied for massively parallel systems, the issues are somewhat different for a distributed system. This work defines a metric for the scalability of distributed systems, based on performance and cost. Using an example for connection management, the usefulness and the intuitive basis for the metric are demonstrated. The proposed metric has a close relationship with the laws of Amdahl and Gustafson for parallel software scalability.

For parallel computation, several different metrics have been defined. They can be used to describe the effectiveness of changing various parameters of the system (e.g. [Sun95], [Grama93], [Sun93], [Sarukkai95] etc.), with respect to given algorithm-machine combinations. Such analysis may be useful (e.g. [Sarukkai95]) for selecting the most scalable algorithm for a given paradigm.

However, distributed systems are usually more complex than the parallel systems. They typically have heterogeneous processors and interconnection mechanisms, an irregular topology of connections (compared to homogeneous processors and communications and a regular topology of connections in most parallel systems), and they handle multiple concurrent inputs. Geographic separation and heterogeneity introduce overhead operations for protocols, security and trading, which change sharply under changes in interconnection and distribution. Merely increasing the hardware resources may not scale up the system's performance if the bottleneck still lies in the software/middleware. Scaling up a software resource affects the overall workload patterns in its clients as well as servers.

This paper defines a scalability metric that relates the *power* of a system to the cost of obtaining that power. It is easy to evaluate and use. This metric could help making decisions about evolving the system along the most economical path. It could also be used for tuning the system for a given set of resources, in order to maximize the concurrency between its hardware and software modules while minimizing the communication and other overhead costs, thus enabling it to

A Scalability Metric for Distributed Computing Applications in Telecommunications

P.P.Jogalekar, C.M.Woodside
Department of Systems and Computer Engineering,
Carleton University, Ottawa, Canada.
tel: 1-613-788-5721, fax: 1-613-788-5727
email: ppj@sce.carleton.ca , cmw@sce.carleton.ca

Abstract

Telecommunications systems must be scalable, meaning that they can be economically deployed in large and small versions. The configurations will differ in their hardware, software, interconnection and traffic parameters. Many future systems will be implemented by distributed software. A new metric is defined to evaluate the scalability of a distributed system from one configuration to another, and it is demonstrated on a connection management example based on the TINA Distributed Processing Environment. The new metric is quite different from metrics for massively parallel systems. It accommodates complex patterns of change and successfully captures intuitive notions of scaleup in a simple formula.

1. Introduction

Distributed software is a key component of modern telecommunication systems. Distributed software applications may include telecommunications services management, customer network management, connection management, fault management, etc. These systems are composed of several clients and servers (or "building blocks") which communicate with each other over a distributed computing environment, typically using remote procedure calls.

This software must be scalable, that is, it must be capable of being deployed efficiently at both large and small scales. Over time it must also be possible to add capacity to either support more users, or enhance the quality of service, or both. If added resources do not increase the scale