Evaluating the Scalability of Distributed Systems

Prasad Jogalekar, Murray Woodside
Dept. of Systems and Computer Engineering
Carleton University,
Ottawa K1S 5B6, Canada
email: {ppj, cmw}@sce.carleton.ca

Abstract

A system design is scalable if it can be economically deployed at a range of scales, in both small and large configurations. Little attention has been paid to measuring and comparing the scalability of different designs of software for distributed operation. Recently a new measure of scalability (called here P-scalability since it based on the "power" metric) has been defined specifically for distributed systems. This paper generalizes the metric, defines a scaling path embodying a strategy modifying the system as it is scaled up, and employs scalability enabling parameters which adapt the system to give the maximum value of the scalability metric, at any point along the scaling path.

1. Introduction

A system is scalable if it can be deployed effectively and economically over a range of different "sizes", suitably defined. It would be useful to be able to evaluate a design for scalability, with a suitable measure. This paper addresses the question of a scalability measure for distributed heterogeneous service systems, such as are used in business computing (client-server and transaction-processing systems), in telecommunications (for intelligent networks and for management of distributed services) and in many otherapplications. It also defines the "size" of the system, through the concept of a scaling path.

Scalability measures have been developed for parallel scientific systems. In [Grama93], [Sun93] *the fixed size speedup* (obtained by scaling the number of processors for a fixed workload) has been defined as follows:

Fixed size speedup =

$$W \cdot C_s \Big/ \left(\sum_{i=1}^{p} (W_i C_p)/i \right) \qquad (1)$$

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:

Fixed time speedup =
<u>(Sum of the work done by all the parallel processors)</u>
(Work done by a sequential system in the same time) (2)

These definitions cannot be used for heterogeneous distributed systems, however. The parallel systems execute a program once and terminate, so the execution time is a natural measure of effectiveness. They are symmetrical and execute the same program on every node, so the number of nodes is a suitable scaling attribute. Distributed systems of the kind we are considering typically run continuously with many simultaneous users and transactions, and are evaluated by their capacity and the quality of service they give to different classes of users.

In [Jogalekar96] the authors evaluated a scalability measure based on the "power" metric of Giessler et al [Giessler78], which is

$$P = (\lambda / T) = Power \qquad (3)$$

where λ is throughput and T is mean delay. They defined a "P-scalability" measure from configuration 1 to configuration 2 to be the quotient of the Power/Cost ratios:

$$\psi_P = [\lambda_2/(C_2 T_2)]/[\lambda_1/(C_1 T_1)] = P-scalability \qquad (4)$$

where λ, C and T are throughput, cost and mean response time in the two configurations.

If configuration 2 handles k times the throughput for k times the cost and an equal response time, it gives ψ_P = 1. This indicates a perfectly "scalable" system. A lower throughput with a lower delay can also give $\psi_P = 1$.

The metric (4) has a fatal weakness however in that it goes to infinity as T goes to zero, while real quality of service measures level out when the delay is "small enough". In this paper the metric is modified to account more realistically for the evaluation of small delays.

Another key idea in describing scalability, which appears to be new, is the notion of a *scaling path* which summarizes a plan for scaleup, controlled by a scale factor. At each scale factor the system is tuned by adjusting scalability enablers, which are parameters that can improve performance. The increase or decrease of the scalablity metric for the tuned system, along the path, evaluates scalability, with a value of unity for "perfect" scalability.

The remainder of the paper defines the new metric, evaluates it on some idealized systems for which the scalability is intuitively obvious, and applies to a real prototype system.

2. The Scalability Metric

A very general family of metrics can be based on the following definition:

$$\psi = F(\lambda_2, QoS_2, C_2) \div (F(\lambda_1, QoS_1, C_1)) \qquad (5)$$

where F evaluates the performance and the economy of operation of the system at two different scales of deployment. The reasoning behind the selection of variables in F is, that:

- λ evaluates the rate of providing valuable services to users, which is related to revenue capability,
- *QoS* is a set of parameters which evaluate the quality of the service seen by users,
- C reflects the cost of providing service.

It seems fundamental that F should be proportional to λ/C, so we concentrate on defining a factor for quality of service. In this paper *QoS* is represented by the mean delay T, although the later developments apply equally to other *QoS* measures such as jitter, or percentile limits on delay. The *QoS* factor will also be based on a target delay $\hat{T}$ and the dimensionless ratio $T/\hat{T}$, and will take the form of a function $f(T/\hat{T})$. This makes:

$$F = (\lambda f(T/\hat{T}))/C \qquad (6)$$

Two possibilities for f are:

(a) $f_a\left(\frac{T}{\hat{T}}\right) = 1/\left(1 + \frac{T}{\hat{T}}\right)$

(b) $f_b\left(\frac{T}{\hat{T}}\right) = 1/\left(1 + \left(\frac{T}{\hat{T}}\right)^n\right)$, n > = 1

and this work will use the version (a). Thus:

$$\psi = \frac{\lambda_2 \cdot C_1 \cdot (T_1 + \hat{T})}{\lambda_1 \cdot C_2 \cdot (T_2 + \hat{T})} \qquad (7)$$

Scaling Strategy

A strategy for scaling up a system will be defined by a *scaling factor k* and a set of *scaling variables* which are functions of k. They express the strategy as a *scaling path* in a space in which they are the coordinates. An artificially constructed example of a scaling path is shown in Figure 1 with three scaling variables Nusers = k, Datasize = $10^4 \log k$, and Nproc = k/100 = number of processors.

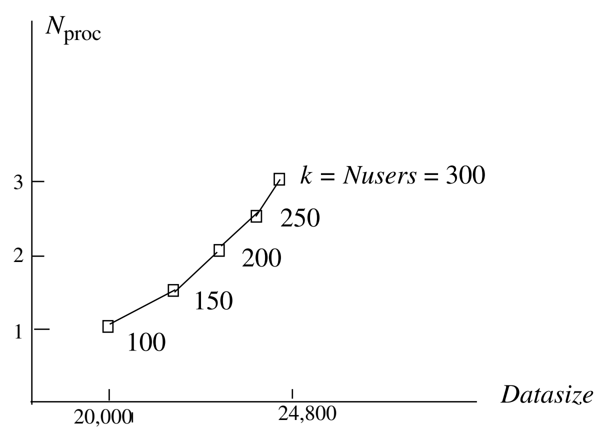

Fig.1 Scaling variables and the scaling path, on an artificial example.

Given a value of k and the resulting values of the scaling variables, other aspects of the system should be adjusted to their best possible values, to make the best of the given choices. Parameters which can be adjusted are part of the scaling plan, and are called *scalability enablers*. The metric used here assumes that optimal values are found for the enablers by optimizing a performance model of the system.

This highlights the importance, in a scalability strategy, of providing powerful enabling variables as tuning parameters in the system. Examples of deliberately provided enablers include:

- multi-threaded software with tunable threading levels (rather than a single thread),
- object migration (rather than fixed allocation of software objects to the processors),
- multiple copies of objects (rather than single copies)
- multi-tasking software designed for a shared-bus multiprocessor, to which processors can be added (rather than a uniprocessor),
- caching the data on the client nodes (rather than no caching)
- tunable buffer sizes (for the remote procedure calls) and thread stack sizes.

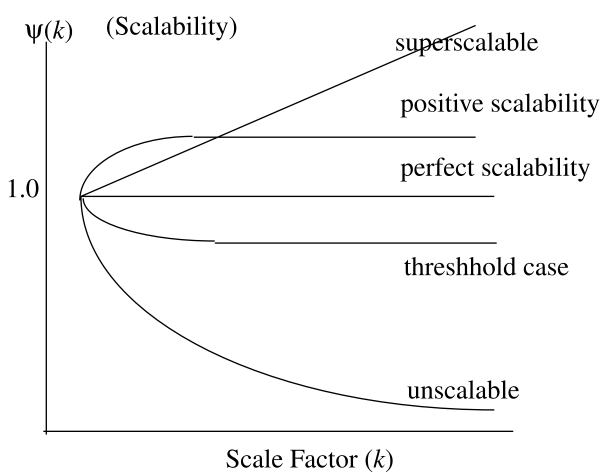

Fig.2 Scaling Behavior

Figure 2 shows how $\psi(k)$ might behave in different situations.

3. Idealized Cases

One demonstration of the "reasonableness" of a scalability metric is to show that it gives an intuitively obvious result for an example which gives a simple solution, and which intuition suggests is, or is not, scalable. This section considers three examples which are scalable, and two which are not.

Idealized Case A: A speed-scaled open system with proportional costs.

Here we consider any system architecture and behavior whatever which handles traffic arriving to it, and which has a steady state. Configuration 2 is uniformly sped-up or time-scaled by a factor k, so the input rate and every component (computers, networks, storage) is faster. The cost is k times greater. Then $\lambda_2 = k\lambda_1$, $T_2 = T_1/k$, $C_2 = kC_1$, and a little manipulation of equation (7) gives:

$$\psi = 1 + \frac{(k-1)T_1}{T_1 + k\hat{T}}, \quad \lim_{k \to \infty} \psi = 1 + \frac{T_1}{\hat{T}}$$

This example has positive scalability with a bounded increase. If $T_1 \approx \hat{T}$ then ψ levels off around a value of 2.

This agrees with intuition.. The throughput increase pays for the extra cost in exact proportion, and the shorter response time provides a bonus which is bounded because f_a is bounded as t goes to 0.

Idealized Case B: Closed idealized system scaled in users and speed.

Consider an idealized system with a separable queueing network performance model with K servers, in which the population N and the (single, constant-rate) server speeds are scaled by k. For simplicity, suppose this is a balanced system with equal demand D sec per response at each of the K servers. Then the throughput and response time are $\lambda = \frac{N}{[(N+K-1)D]}$, and $T = \frac{N}{\lambda} = (N+K-1)D$. Intuitively, this system should give perfect scalability and a value of ψ near to unity.

Substituting into (7) with $N_2 = kN_1$, $D_2 = D_1/k$ and $C_2 = kC_1$ we get

$$\psi = \frac{k(N_1 + K - 1)}{(kN_1 + K - 1)} \cdot \frac{(N_1 + K - 1)D_1 + \hat{T}}{(kN_1 + K - 1)\left(\frac{D_1}{k}\right) + \hat{T}}$$

$$\lim_{k \to \infty} \psi = \frac{(N_1 + K - 1)[(N_1 + K - 1)D_1 + \hat{T}]}{N_1[N_1 D_1 + \hat{T}]}$$

By inspection this limit is also greater than 1.0, but if $K << N_1$ the limit approaches 1.0.

If the system is not balanced then there will be a bottleneck server (having the largest value of D). In this case as k increases, N becomes large, $\lambda \to 1/D_{max}$ because of the bottleneck. As the servers are all scaled up, $D_{max,2} = D_{max,1}/k$, we have $\lambda_2 = k\lambda_1$ and thus $T_2 = N_2/\lambda_2 = N_1/\lambda_1 = T_1$. Thus for large enough k, $\psi \to 1.0$ just as in the balanced case with large N_1.

Idealized Case C: Closed idealized system with replication of servers.

As in Example B, this system has a separable queueing model. To scale it up, each server is replicated k times and its load is divided equally among the k servers. There is no overhead cost associated with the replicas (a major idealization). Again consider a balanced version with all demands D equal; then:

$$N_2 = kN_1;\ K_2 = kK_1;\ D_2 = D_1/k;\ C_2 = kC_1$$

Following Example B, but with $K_2 = kK_1$ we obtain almost the same limiting ψ:

$$\lim_{k \to \infty} \psi = \frac{(N_1 + K - 1)[(N_1 + K - 1)D_1 + \hat{T}]}{N_1[(N_1 + K)D_1 + \hat{T}]}$$

and again if $K << N_1$, $\psi \to 1.0$. For an unbalanced system the result is also the same.

As in the previous two cases, intuition again suggests that this is a perfectly scalable system, which agrees with the result for the measure.

Idealized Case D: System with a single bottleneck.

The usual way to identify non-scalability is to discover a single bottleneck which cannot be speeded up. The effect of such a bottleneck on ψ is straightforward; it limits λ to some value λ_{max}. In a closed separable system it is easy to see that $T \to N/\lambda_{max}$, so if $N_2 = kN_1$ and $C_2 = kC_1$, then for large k:

$$\psi \approx \frac{\lambda_{max}}{\lambda_1} \cdot \frac{C_1}{kC_1} \cdot \frac{N_1/\lambda_1 + \hat{T}}{kN_1/\lambda_{max} + \hat{T}}$$

$$\psi \to \left(\frac{\lambda_{max}}{\lambda_1}\right)^2 \frac{1}{k^2}$$

This gives a sharp decline towards zero which clearly identifies the non-scalability of the example.

Idealized Case E: System with Replication and Overhead.

In Example C, suppose that for $k>1$ the demands D are augmented by coordination overhead, for example to maintain overall system state data. An efficient coordination mechanism might limit this to a slowly increasing amount of overhead, giving a relationship such as

$$D_2 = (D_1/k) + D_0 \log k$$

Then from (7),

$$\psi = \frac{k(N_1 + K_1 - 1)}{(kN_1 - K_1 - 1)} \cdot \frac{(N_1 + K_1 - 1)D_1 + \hat{T}}{(kN_1 + kN_1 - 1)\left(\frac{D_1}{k} + D_0 \log k\right) + \hat{T}}$$

For large k,

$$\psi \to \frac{(N_1 + K_1 - 1)[(N_1 + K_1 - 1)D_1 + \hat{T}]}{N_1(N_1 + K_1)D_0 \log k}$$

which decreases slowly towards zero, especially for large k. A "large" value of k implies

$$kD_0 \log k > D_1$$

$$k \log k > D_1/D_0$$

Then if D_0, which describes the relative magnitude of the coordination overhead, is 10% of D_1 this limit would be passed at $k\log k = 10$, which (if logarithms are to base 10) is $k = 10$. Then for $k<10$ the system would be "somewhat" scalable.

4. Example and Scalability Analysis

A case study was done on a prototype system for managing connections for multipoint conferences over an ATM network, with the processes and interactions shown in Figure 3. The Users request PA to reserve bandwidth for multipoint sessions, and to free up resources. PA authenticates the request and passes it to NRM and CM. NRM uses RM to determine routes and SM to set up physical connections in subnetworks, and it updates DB which maintains the state of all network resources. CM establishes and releases virtual connections between end user sites using RM and SM. The “entries” marked E1, E2, E3 in the diagram are access points to different services by each task.

It is far from obvious what the scalability limits of this system are. There are questions posed by many potential bottlenecks at the various servers, and by providing sufficient threads at the servers to avoid starvation of the processors.

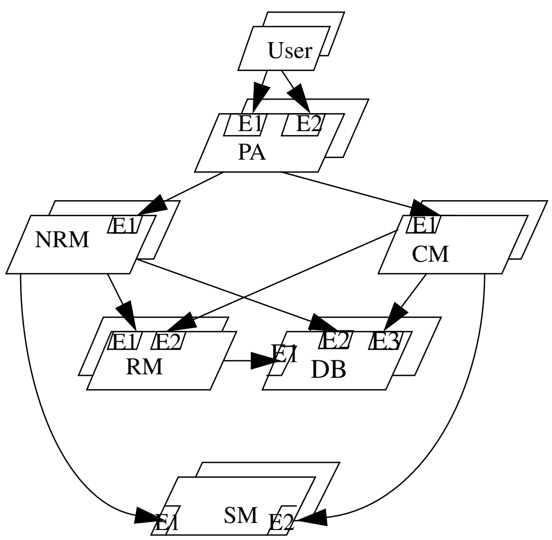

Fig.3 Layered queueing model of the example connection management system

The scaling path for this system was based on adding one processor at a time, starting from a base system with four processors. To limit the effort, the path was only carried to seven processors. The scalability enablers which were optimized at each scale were the number of copies of the database DB, the allocation of tasks to processors, the number of threads used in multi-threaded servers, and the number of users being served. This scaling path is referred to below as path number 3, and it was compared to two relatively rigid scaling paths (1 and 2) that had fixed decisions on the database and the allocation.

The performance evaluation was made using an analytic performance model, which was a layered queueing model implemented in the solver LQNS [Franks95]. The workload parameters of the base case are given in Tables 1 and 2. These are not the values of the actual industrial prototype system, but were invented for publication. However they show the principle of operation of the analysis just as well as the measured values would. Parameters are associated with entries, as labelled on Figure 4, which are identified with the services offered by each process.

The scalability metric given by equation (7) was used, with the nominal value of response time, $\hat{T} = 150$ sec. This “good enough” value of response time may seem to be long, but it is the time to set up a multipoint conference with many connections. The cost C was taken to consist of one unit of cost per processor and 0.1 units per copy of the database (the licensing cost).

Table 1: Mean Execution Times of the Entries

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

Evaluating the three scaling strategies

When k increases by one, each scaling strategy adds a processor to the previous configuration. Strategy 1 always re-allocates the single most-utilized task to the new processor, and creates a new database replica on the previously least utilized processor. It then determines the optimum number of users by a golden-section search. Strategy 2 also adds a database replica and then re-allocates all the tasks, in round-robin fashion in the decreasing order of total utilization. Finally it optimizes the number of users. Strategy 3 performs an optimal allocation over the enlarged set of processors using simulated annealing (see for instance [Ingber93] for a description of simulated annealing). The optimization includes an optional additional database replica, the number of threads at each server and the number of users. Simulated annealing was used without special tuning for the application, and has not necessarily found the global optimum every time; it represents a "good try" based on the use of standard methods.

Because it has more enablers, the last strategy is expected to give the best scalability. However the relative capability of the simpler, more fixed strategies is also of practical interest.

Results

The results for scaling factors from 4 to 7 (i.e. 4 to 7 processors,) are shown in Figure 4. Strategy 2 is the worst, and strategy 3 is the best as expected. Strategy 3 even achieves a measure greater than unity for k = 5, 6 and 7. Beyond this range, the way the curve is dropping suggests there is a single bottleneck which will limit further scaleup.

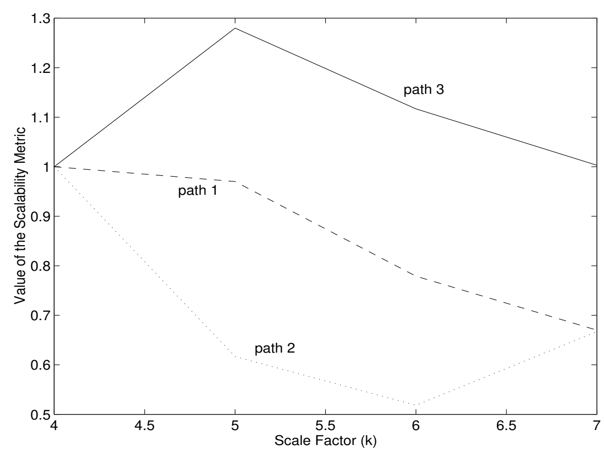

Fig.4 Evaluating different scaling paths

Table 3. Performance figures at maximum values of the scalability metric

	Path 1			Path 2			Path 3		
	N	R(sec.)	$\lambda(sec^{-1})$	N	R(sec.)	$\lambda(sec^{-1})$	N	R(sec.)	$\lambda(sec^{-1})$
k = 4 (ref.)	5	124.7	0.02707	5	124.7	0.02707	5	124.7	0.02707
k = 5	7	130.3	0.02341	5	153.6	0.02341	12	171.8	0.05177
k = 6	7	142.7	0.03453	5	150.4	0.02366	12	165.1	0.05330
k = 7	7	142.7	0.03453	7	143.6	0.03437	12	159.9	0.05456

To provide a fair comparison, the number of users in the reference configuration was optimized at 5 users, giving a response time of 124.7 seconds and throughput of 0.02707 per second, as shown in the first row of table 3.

In case of scaling path 1, due to the arbitrary and not very intelligent strategy for migrating objects, some CPUs were not utilized at all. At $k = 6$, one CPU is not used and at $k = 7$, two CPUs are not used. The cost of the unused CPUs reduces the value of the metric.

For scaling path 2, the round-robin allocation only takes into account the execution demands and not the communication frequency between tasks. Clearly, the scaling path poorly handles the bottlenecks as new resources are added, and performs poorly as expected. At $k = 7$, this strategy happens to do better load balancing by moving the CM task, which is the bottleneck, to a newly added CPU. This seems to be a happy accident.

The advantages of using additional sclability enablers and an automated optimization are clear in the success of path 3. The optimization finds a significantly better allocation than in the reference case, and is able to greatly increase the throughput and the number of users. The increased response time is still near the target $\hat{T} = 150$ seconds. If this range of scales is adequate, then we do not need to reengineer the system, but can deploy these versions over 4 to 7 processors.

This experiment was limited for a small range of scales, but the method extends to large systems in a straightforward way. The scaling path 3 is still rather limited in its potential to provide large scale service. More tasks need to be replicated to avoid becoming bottlenecks. The number of replicas should be adjusted, as we have done through the database server.

5. Conclusions

A general family of scalability measures has been proposed and has been tested by applying it to a variety of intuitively scalable (and non-scalable) abstract models of systems, and by analyzing a real system with various scaling strategies. The measure is related to economic and quality-of-service factors, and appears to reflect intuitive ideas of scale-up quite well.

The scalability metric defined here gives reasonable results on the various idealized types of large systems. It was also effective for evaluating scaling strategies and designing a good scaling plan for a real application. It was much better than a metric based on the power function in giving credible solutions. The power function unduly rewards very small response times.

The example showed how quite a simple scaling strategy may be sufficient, and may be shown to be sufficient over a limited range of scales.

The notion of scaling strategy and a scaling path were introduced. Such a strategy allows tuning the “scaling enablers” effectively, such that the resulting value of the scalability metric is not much less than unity.

The notion of “scaling enablers” embedded in a strategy captures those features of the system’s architecture, which can be adjusted in order to enhance the scalability of the system. Scalability analysis, as proposed in this paper, can be used (1) to make a strategic choice of a scaling strategy, among several alternatives, (2) to tune the available scaling enablers for achieving the most cost-effective system configuration at any given scale, (3) to assess the need for providing more scaling enablers or (4) to assess the need to re-engineer the system, given the available choices for an expected evolution of the system.

The simple scalability metric discussed at length here is a function of mean values of performance attributes. Clearly, it would also be beneficial in some systems to define a measure that uses more general QoS attributes, such as percentiles of response time, probability of missed deadlines, or response time variance. There may also be many types of responses, in which case, the measure should be a combined function of their QoS.

The “scaling factor” k, which defines the degree of scale-up in this work, is a scalar. There are many advantages of having a scalar factor, and it is a familiar concept. For instance. today’s scalable benchmarks (such as the TPC benchmarks) are based on a single scale factor. However some situations may demand two or more independent scale factors (e.g. a factor for the volume of requests handled by a database management system, and another for the database size). In such cases, a vector scale factor could be used.

Acknowledgments:

We would like to thank Pankaj Garg, Jerry Rolia, Fahim Sheikh, John Omura and Roger Hill for their comments on this research. This research was supported by the Natural Sciences and Engineering Research Council of Canada through its program of Industrial Research Chairs.

6. References

[Franks95] R.G.Franks, A.Hubbard, S.Majumdar, J.E.Neilson, D.C.Petriu, J.Rolia and C.M.Woodside, *A Toolset for Performance Engineering and Software Design of Client-Server Systems*, Performance Evaluation, vol. 24, no. 1-2, Nov. 1995, pp. 117-135.

[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.

[Ingber93] L.Ingber, *Simulated Annealing: Practice Versus Theory*, J Mathl. Comput. Modelling, Vol. 18, No. 11, Dec. 1993.

[Jogalekar96] P.P.Jogalekar and C.M.Woodside, *A Scalability Metric for Distributed Computing Applications in Telecommunications*, SCE-96-07, Department of Systems and Computer Enginering, Ottawa, Canada, submitted to the 15 th International Teletraffic Congress, 1997.

[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.