HEWLETT PACKARD

Design-Time Simulation of a Large-Scale, Distributed Object System

Svend Frølund, Pankaj Garg
Software Technology Laboratory
HPL-98-158
September, 1998

scalability analysis, performance modeling

We present a case study in using simulation at design time to predict the performance and scalability properties of a large-scale distributed object system. The system, called Consul, is a network management system that is designed to support hundreds of operators managing millions of network devices. It is essential that a system such as Consul be designed with performance and scalability in mind, but due to Consul's complexity and scale, it is hard to reason about performance and scalability using ad-hoc techniques. We built a simulation of Consul's design to guide the design process by enabling performance and scalability analysis of various design alternatives. A major challenge in doing design-time simulation is that many parameters for the simulation are based on estimates rather than measurements. We developed analysis methods that derive conclusions that are valid in the presence of estimation errors. In this paper we describe our *scalability analysis* method for design simulations of distributed object systems. The main idea is to use *relative* and *comparative* reasoning to analyze design alternatives and compare transaction behaviors. We demonstrate the analysis approach by describing its application to Consul.

1 Introduction

Superior performance and scalability (P&S) is one promise of distributed computing. Scalability implies that a distributed solution can accommodate growth in user demands while providing a stable Quality of Service. Currently, there is no readily-applicable, systematic approach for designers of distributed software systems to understand the P&S characteristics of their designs. P&S characteristics of distributed system designs are hard to predict and test because: (1) a typical distributed application executes multiple transactions concurrently and asynchronously, (2) network delays are non-deterministic, and (3) testing and measurement require expensive deployment. This can lead to costly software life-cycles, if P&S bottlenecks are discovered at deployment times and systems have to be re-designed and re-implemented.

We present a case study in using simulation, at design time, to predict and analyze the P&S of large-scale, distributed software systems. The case study involves simulation of a distributed network management system, called **Consul**, being developed by the Hewlett-Packard Company[1]. The simulator, **Damson**, can be used to predict the behavior of Consul under various workloads and configurations.

Design time P&S analysis is challenging because many of the model parameters, such as resource demands, are based on estimates rather than measurements. We want the P&S analysis results to be valid in the presence of such input estimation errors. For this reason we derive results that are relative rather than absolute in nature. For example, we can compare two designs to determine which is more scalable; we cannot, with as much confidence, draw conclusions about the absolute scalability of an individual design.

We introduce a P&S analysis framework that can be used for design time simulation models to derive relative results about P&S characteristics. We apply the analysis framework to Damson, and describe the results that came out of that analysis of Damson, and discuss the lessons learned in the process.

The rest of the paper is structured as follows. In Section 2 we describe related work in the area of performance and scalability analysis. In Section 3 we describe the architecture and functionality of Consul. In Section 4 we describe the Damson simulator of Consul, built using the SES/Workbench tool [5]. In Section 5 we explain our scalability analysis technique, and in Section 6 we describe its application to Damson. Finally, Section 7 summarizes our major findings and lessons learned.

2 Related Work

The main distinction of our case-study is the performance and scalability evaluation of a realistic, industrial distributed object system design. The techniques used in this study have benefited from the literature on systems performance evaluation (e.g., [1, 2]). In this section we describe how our work relates to some published work in this area.

A performance model based approach has been applied for engineering of several distributed systems, e.g., file systems [16, 17, 18, 19], and databases [20, 21]. In all these applications, the

1. Disclaimer: Although we present some information on the design of Consul in this paper, this should not be construed in any way as a promise by Hewlett-Packard Company to deliver a product based on this design.

performance model and its analysis is calibrated and validated against system measurements. This makes such performance studies different from our study. First, the performance analysis of the kind reported in these publications require portions of the system under study to be implemented and available for measurements—a non-trivial departure from providing performance guidance at the early stages of the software lifecycle. Secondly, the expense of such performance studies makes them cost-effective only for systems software that will have large-scale general-purpose use, e.g., systems software. For application programmers with tight deadlines and schedules, such performance studies are impractical.

Early work at the University of Texas [15] emphasized the need for design-time performance study of distributed applications. Not surprisingly, the tool that we have used to develop our performance model (SES [5]) is derived from this early work. We have not found, however, any publication (from the same group or the SES vendor) describing a industrial-strength case-study. Moreover, as with some of the other *analytical* performance evaluation studies of distributed application designs (e.g., [3, 25]) the focus seems to be on client-server computing. Although a useful step towards performance evaluation of distributed systems, such analysis does not readily apply to distributed object systems. The main drawback is that the server in client-server systems is not allowed to make requests of its own to other servers.

Researchers at University of Toronto [6] and Carleton University [4, 22] are developing layered queueing network (LQN) modeling techniques to overcome this limitation. In a parallel study, we have applied the LQN approach to the design reported in this paper [7]. We are working on incorporating the LQN methods for performance and scalability analysis of large scale distributed object systems. LQN abstractions are high enough to facilitate rapid model development, but at the same time they are much farther from a design than a simulation model. We are now looking at an hybrid approach that will allow us to smoothly and automatically shift from simulation and analytical models, and vice versa, over the course of the software life cycle. Smith's work on Software Performance Engineering [23, 24] is related to our modeling and analysis methods. Her work so far, however, has focused on sequential software systems. Moreover, her design-time analysis methods are quite different from the scalability analysis method developed in this paper.

3 Consul: A Distributed Network Management System

Consul's application domain is network management [8]. Network management involves monitoring the operation of a computer network in order to detect device failures, determine device load, and detect link failures. A network typically contains a wide variety of devices: computers, printers, and routers. A network management system supports human operators in managing a network, it provides various views of the network state and connectivity, and allows the operators to perform queries over these views and initiate actions to change the network state.

Consul is an object-oriented framework for creating network management applications. Consul implements a number of services that are commonly found in network management systems. Different network management applications can customize and reuse the Consul services, and integrate them with application-specific services and legacy components. It is possible to build a completely functional, although generic, management application exclusively from the Consul services without adding application-specific components. Such a generic Consul application is the subject of our scalability case study.

Consul is based on CORBA, which is a standard for communication in distributed object systems. The Consul system represents the managed devices, e.g. routers, hubs, and computers, as CORBA objects. The Consul management services then access and manipulate these objects, and the objects in turn delegate the access and manipulation to the real device in the network. This delegation involves whichever communication protocol that the device supports, e.g, SNMP and CMIP. One of the main advantages of representing devices as objects in Consul is that this communication protocol is hidden from the services; the services access the managed devices through a standard CORBA interface that is independent of the protocol supported by the actual device. We refer to the objects that represent managed devices, as the managed objects.

At runtime, the Consul services are implemented by operating system processes. There may be multiple instances of a given service to increase the capacity of the system and to allow for data replication and partitioning. Figure 1 depicts a typical runtime deployment of a Consul-based

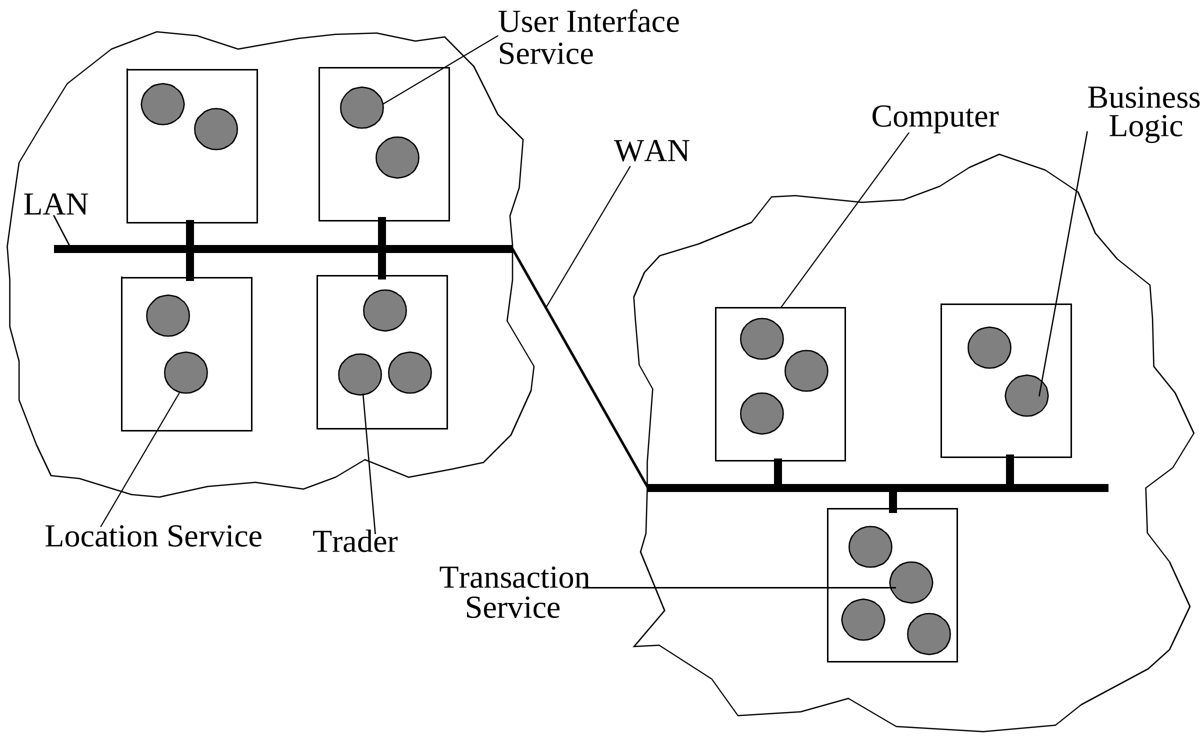

Figure 1. A typical Consul deployment

application. In this illustration, the Consul services are deployed over two local area networks (LANs) connected by a wide area network (WAN). The deployment represents a situation where a network is being managed by multiple management centers and where these centers are connected by a WAN. The services in one center can communicate with the services in the other center through the WAN connection. The communication between the services is mediated by a CORBA Object Request Broker (ORB). The communication is synchronous as well as asynchronous. Each service process is multi-threaded, and multiple requests may be handled concurrently by the same process.

In terms of scalability, the design goals of Consul was to support hundreds of operators managing millions of devices. It is complicated to determine whether these scalability goals can be reached by a particular design, for example, whether the design contains performance bottlenecks that prevents scalability beyond a certain point. Usually, designers have an understanding of the local scalability and performance properties of individual operations such as locating an object at an object server, updating an object attribute, redrawing an object structure on a screen. The challenge is to develop a design-time understanding of the overall scalability of an entire Consul-based management application. Determining the overall scalability is challenging because of the scale and complexity of a system like Consul. Consul contains more than 20 different services that may each give rise to multiple instances in a Consul deployment. Most of the services are multi-threaded, communicate using asynchronous events and synchronous remote procedure calls (RPC), and interact using nested RPCs, upcalls, and forwarding. In the next section we describe the approach that we used to systematically predict the performance and scalability of Consul.

4 Damson: A Simulation Model of Consul

In order to predict the performance and scalability characteristics of different Consul designs, we built Damson, a discrete-event simulation model of Consul. We built Damson at design time, that is, while Consul was being designed. The motivation behind Damson was to allow early detection of scalability limitations and performance bottlenecks in the Consul design, thereby enabling a software architecture that was designed to be scalable and deliver the intended performance.

4.1 Modeling Goals and Requirements

The primary performance metric for Consul is the end-to-end response time that operators experience. An operator's request typically results in activities at multiple services, for example, a user interface service may invoke multiple management logic services that each may access multiple managed objects at object servers. A *transaction* is the total (distributed) activity created in response to an operator request. The starting point of a transaction is an operator request, and the end point of a transaction is presenting the result of the request to the operator.

The Damson model simulates the execution of three types of transactions:

- **T1** captures a query issued by an operator to locate and display the managed objects that satisfy a particular constraint
- **T2** is initiated by an operator and displays the attributes of a particular managed object
- **T3** is initiated relative to an on-screen symbol that represents a map of managed objects; T3 expands the symbol and displays the map.

These are the most important transactions in Consul. Based on past experience, these transactions constitute the bulk of the workload, and they are the most time critical.

With Damson, we want to predict the response time of these transactions under various workloads, configurations, and environments. The notion of workload captures transaction generation by operators as well as the number of devices that must be managed by the system. A configuration is a particular way to deploy the Consul services on a given set of resources (computers, networks, etc.). The notion of environment captures the resource usage by other applications that are deployed on the same resources. To analyze the scalability of Consul, we want to determine how the response time depends on increases in workload and available resources. For example, one dimension of scalability is how transaction response times increase for an increasing number of managed devices.

4.2 Modeling Paradigm

Because our goal is to predict transaction response times, our model focuses on aspects of Consul that cause transactions to spend time. Transactions spend time when they consume and contend for *resources*. Hence, we simulate where and how transactions consume and contend for the following *physical* resources: disk, memory, CPU, and network bandwidth.

It takes time for transactions to consume and contend for these physical resources. For example, it takes a certain amount of time for a transaction to consume the necessary CPU resources to execute a given number of instructions at a process. Moreover, the transaction may be contending for the CPU of a given computer with other transactions. The contention for CPU resources may cause a transaction to spend time in the “run queue” of an operating system.

Moreover, we model contention for *logical* resources, in addition to the above physical resources. For example, with multithreaded servers, transactions may contend for locks to shared data structures. We simulate contention, and the associated queueing, for software locks because it may be a significant component of transaction execution time.

4.3 Structure of the Damson Model

The Damson model was built using the SES/Workbench modeling environment. Figure 2 abstractly illustrates the overall structure of Damson. The Consul services are represented as processes in the Damson model. Although the main goal for Damson was to analyze performance and scalability issues for Consul, the structure of Damson is a *general-purpose structure* for building performance and scalability models of distributed applications. The generality of this structure has been demonstrated by its applicability to model an internet banking application, an internet Web server, and a scalable object access mechanism for CORBA objects. Each of these models was constructed in less than a week! Without this structure, it could easily take a month or two to construct a basic queueing network model simulation of these applications.

Modularity is a key aspect of Damson’s structure. Damson’s structure maintains a clean separation between the following modules: workload, application, environment, configuration, and system. These modules can be readily reused for modeling other distributed applications by defining the application, workload and environment modules, and customizing the system and configuration modules. In the rest of this sub-section we first describe the general aspects of these modules, and then illustrate their use with explanations of how they are used to model the Consul design.

Application module: it models the behavior of the various application components, such as processes, transactions, and communication mechanisms. SES/Workbench provides a wide variety of modeling abstractions for modeling the behavior of a distributed application [13]: *processes* are modeled as SES/Workbench *submodel types* through which *transactions* flow. Each submodel

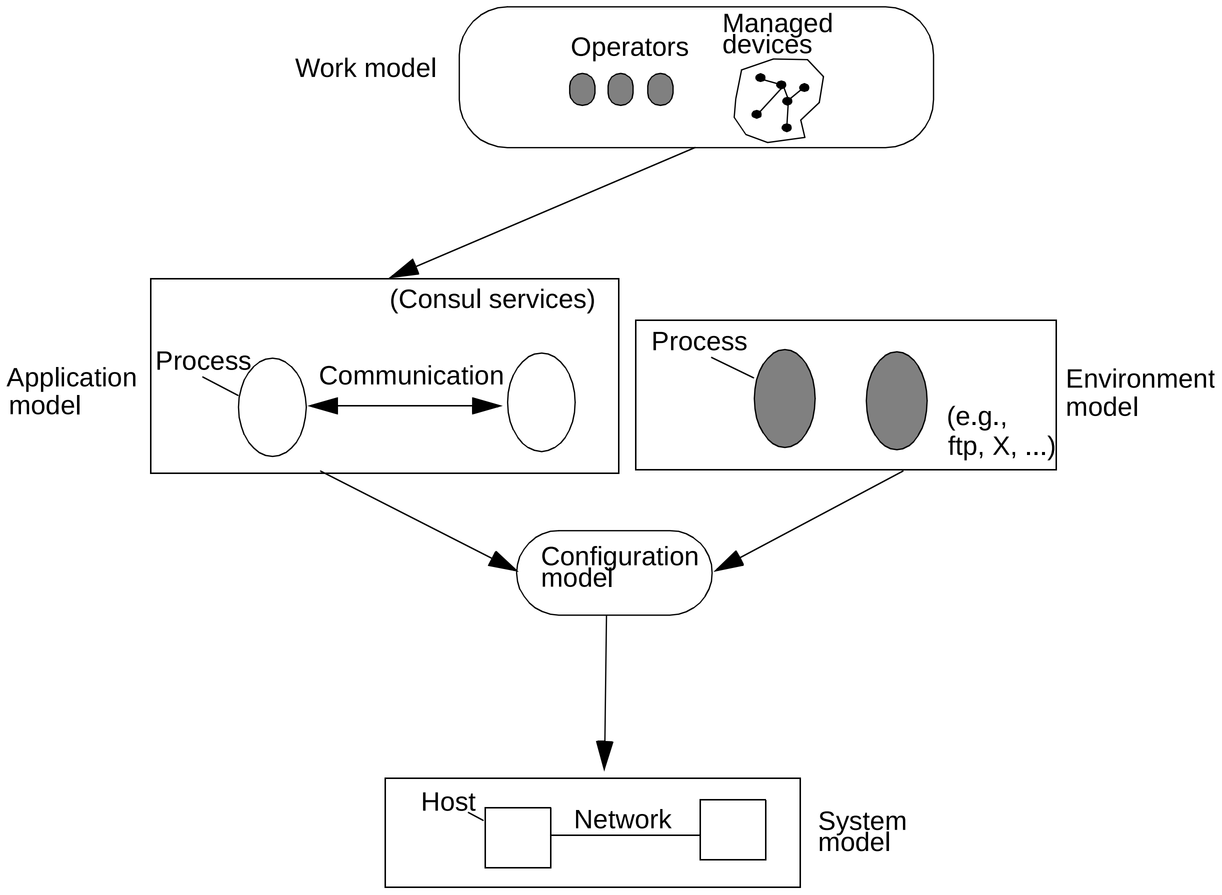

Figure 2. The abstract structure of Damson, a performance and scalability model of Consul

type can have several instances, which allows us to model multiple instances of the same process. While flowing through a process, a transaction may consume resources (e.g., disks, CPU, network). A process can *communicate* with another process by sending a transaction to the other process. Such inter-process communication can be asynchronous or synchronous. A process can synchronize with another process by waiting on a condition variable. A process can *interrupt* another process. A transaction can store local state information, e.g., number of disk bytes to be read. Processes can create new transactions to model individual threads of computation. Contention between threads can be modeled using queueing servers. The control flow inside a process can be either deterministic or probabilistic.

System module: it models the hardware (computers and networks) and system software (operating system, middleware, and networking software) on which the application components are deployed. We model CPUs and disks as basic queueing servers.We model the network as a delay server, where the delay is based on the *latency* and *bandwidth* of the network. Figure 3 illustrates our model of the network. Each process is configured to be running on a host. Each host, in turn, is

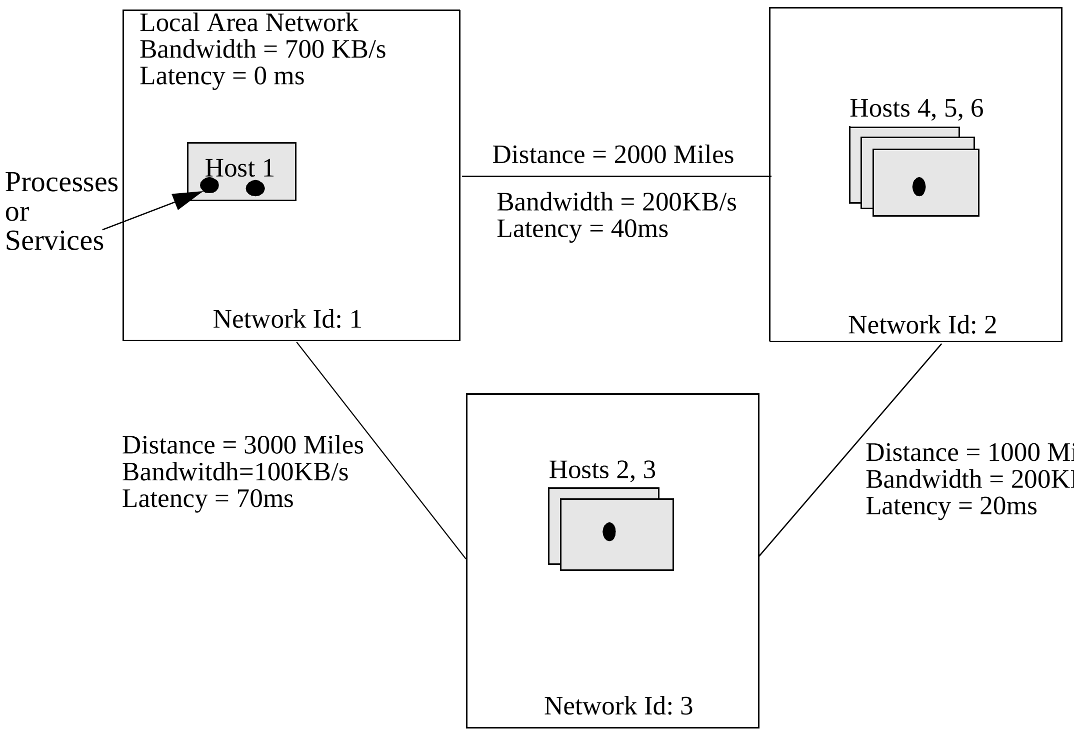

Figure 3. Network model in Damson.

configured to be in a particular local area network. Finally, different local area networks are connected to each other by wide area networks. We assume all the local area networks to have the same bandwidth and latency parameter values. The bandwidth and latency of wide area networks is assumed to be dependent on distance. All the timings are Damson input parameters, so they can easily be changed for different settings.

Part of the system module is a model of a CORBA object request broker (ORB). An ORB mediates RPC calls between services. We model a call through the ORB as making service demands on the CPU of both the sending and receiving hosts and the underlying communication network. The amount of CPU time on the initiating host is assumed to be a linear function of the message size.

Environment module: it models the behavior of other applications that are deployed on the same hardware. The environment module is similar in structure to the application module.

Configuration module: it models the “mapping” of application and environment components to system components. This mapping is accomplished by associating a “host ID” variable with each modeled process.

Workload module: it models the entities that generate work or influence the work complexity for the model. In the case of Damson, work is modeled as operator requests that result in transactions. The devices being managed by Consul affect the work complexity, i.e., the amount of resources consumed by transactions.

The key to the *generality* of the structure of Damson is that each new application modeled can reuse the system module and the underlying concepts and structure of the other modules. For example, the Consul-specific aspects of Damson are as follows.

1. **Application module—**models the different Consul services and their interactions with each other. We model eleven services. Each service is modeled as a multi-threaded server, and each service can be processing more than one request at any given time.
2. **Workload module—**models the workload from Consul operators. We assume a number of operators, each generating transactions with a Poisson process with a fixed mean interarrival rate. An operator's requests are routed to be either a T1, T2 or T3 transaction with equal probability. In a T1 transaction, an operator issues a query to find devices that satisfy a set of properties. In a T2 transaction, an operator requests more information about a particular device by clicking on the symbol representing that device. In a T3 transaction, an operator requests a topological map of a network of devices. The execution of T1 and T3 transactions involves a number of managed objects. When we instantiate T1 and T3 transactions in the simulator, we assign to them a uniformly distributed number that represents the number of objects involved. The ranges for these uniform distributions are given as input parameters.
3. **Environment module—**creates environmental processes (processes other than Consul processes) on the hosts that Consul is deployed on (e.g., ftp daemon). These processes consume CPU and disk on the host, at regular (deterministic) time intervals.

4.4 Model Execution

To run the Damson model, we must provide values for all the model parameters. These parameters capture a variety of properties of a Consul deployment:

- Resource demands by service methods. These parameters reflect the resource (such as CPU, disk, and memory) that a transaction consumes when executing a given method in a given service process. At design time, these service demands cannot be measured, but must instead be estimated.
- Transaction branch probabilities. These parameters provide values to perform a stochastic choice between multiple execution paths for transactions. Like resource demands, these parameters are also based on estimates rather than measurement.
- System configuration. These parameters describe the physical resources, such as computers and networks, on which the deployment takes place.
- Service configuration. These parameters describe how many services are instantiated, how these services are mapped to specific computers, and how the service processes themselves are interconnected.
- Environment configuration. These parameters capture the number of environmental processes, and how these processes are mapped to specific hosts.
- Workload. These parameters determine the number of operators, and they describe how operators generate transaction in terms of frequency and think time. The workload parameters also determine how many devices the deployment must manage.

One of the main issues with design time simulation is that parameters, such as resource demands and transaction branch probabilities, must be based on estimates rather than measurement. As we shall see in Section 5, such estimated resource demands give rise to derivation of relative, rather than absolute, performance and scalability characteristics, during analysis of the model output.

We instrument the simulation model to measure the end-to-end response time of transactions in the model, and we use this simulated response time as an estimate or prediction of the expected "real" response time. In order to obtain statistically "good" estimates of the *mean estimated transaction response time* (MERT), we run the model for a long duration (for transient removal), and use batch means to estimate errors in the MERT due to stochastic variations.

5 Scalability Analysis

A design-time performance model such as Damson enables performance estimation of a distributed application under extreme conditions, e.g., what happens to transaction response times when Consul manages about a million devices, how does the response time change when thousands of customers start accessing an internet banking application? Such extreme conditions are critical for successful operations of a distributed application, but are hard and expensive to test in practice. In general, we can define a class of questions that relate to how the application design *scales* across some given dimension; the dimension can be the number of managed devices, the number of clients accessing a system, and so forth. Ideally, we want our designs to remain stable across large variations in the scale dimension, and still guarantee a *Quality of Service* (QoS) by tuning some "knobs" or parameters of the design, implementation, or deployment. The challenge, however, is: what can we say about the scalability of a design, without testing or deploying an implementation?

In the rest of this section we develop a statistics-based scalability analysis technique for design-time performance models. We demonstrate the technique by applying it to Damson, giving insights into the scalability characteristics of Consul. The primary issue in developing these techniques was to enable design-time analysis with weak assumptions about the model parameter estimates, since we do not expect design-time analyses to have many measurement-based parameters.

5.1 Terminology

During the course of this experiment, we found that some basic terms for scalability analysis are not well-defined in the literature, and different people associate different meanings for terms like scalability. Therefore, in this subsection we define scalability, scale factor, scalability point, scalability tolerance, scalability limits, scalability parameters and scalability enablers. Our definitions assume a context in which there is a distributed application design and a corresponding performance model.

Scalability: a distributed software design, D, is said to be *scalable* if its performance model predicts that there are possible deployment and implementation configurations of D that would meet the QoS expected by the end user, within the *scalability tolerance*, over a range of *scale factor* variations, within the *scalability limits*. For example, we can say that the design of a distributed network management application is scalable, if the model predicts that the expected mean transaction response times do not degrade more than 50% for up to a million managed devices. Here the QoS is the end-user response time, the scale factor is the number of managed devices, and the scalability tolerance is 50% variation. One scalability limit is to consider variations of the scale factor for only up to a million managed devices. There is no specification for the costs limit that we are willing to pay for the scalability. Note that for design-time analysis, we specify scalability with reference to degradation or variation in QoS rather than an absolute value.

Scale Factor: the concept of a scale factor captures the idea that the scalability of a design is always with respect to some dimension of the input vector. For example, in the network management applications, a scale factor is the number of managed devices. For an internet banking application, a scale factor is the number of system clients. For a web server, a scale factor is the number of hits per second. A scale factor is a discrete or continuous variable *S*, such that

(1) If *S* is discrete, then *S* can take values $s_1, s_2, \ldots, s_N$ such that

(a) each S_i corresponds to a possible input value for D

(b) for each S_i, $1 < i < N$, the end-user expects the QoS to be within the *scalability tolerance*;

(2) if S is continuous, then $\forall S \subset [S_1, S_N]$ the end-user expects the QoS within the *scalability tolerance*.

Scalability Point: intuitively, a scalability point is a value of the scale factor which is important for the end-user, and at which point we may expect a significant change in the performance of the application. Hence, we define a scalability point as a value S_p of the scale factor S, such that the configuration of the model of D is different for S_p than for all $S < S_p$. From a capacity planning viewpoint, these are the points at which we may need to change the capacity of the hardware or software resources. For example, a trading bank can decide, based on model predictions, that up to a thousand customers can be supported by one back-end CPU, but for more than a thousand customers they must add another CPU to provide the expected response time. Usually these points will coincide with the “knee” of a performance curve.

Scalability Tolerance: the permitted variations in the QoS over the scale factor variation. Normally, we would expect the performance of a distributed application to degrade with an increase in the scale factor. The scalability tolerance allows an end-user to specify the degradation types permitted, and their magnitudes. For example, in a database query operation, a scalability tolerance can be that the search time for tuples increases logarithmically with an increase in the number of tuples. In the case of an internet banking application, a tolerance can be that response time for transactions change by about 50% at peak loads. Since this is a design-time specification and analysis, we expect the tolerance to be specified loosely in terms of relative comparisons of two or more QoS specifications, rather than as an absolute value.

Scalability Limits: they specify the ranges for the costs and scale factor variations that must be considered for scalability analysis. For example, for a distributed network managed software, the designers may consider the scalability of their designs only for thousands to million managed devices. They may not consider a billion managed devices. A telephone operating company may, however, consider managing a billion customers. Similarly, designers can specify the permitted cost growths with the scale factor variations. Sometimes a linear cost increase may be permitted; at other times, a sub-linear cost growth may be more desirable.

Scaling Parameters: the performance model parameters that must be changed at each scalability point. These are the design variables that depend on the scale factor. For example, in the network management case, as the number of managed devices increase, we expect the number of entities in the databases to also increase. This increase will have an impact on the service demand for the database search and update algorithms. Similarly, in the case of internet banking, when the number of clients increase, we expect the database sizes to grow proportionally.

Scaling Enablers: entities of design, implementation, or deployment that can be changed to enable scalability of the design. These are the "knobs" that can be tuned, at the times of design, implementation, or deployment. For example, at design-time a designer can decide to use larger chunks of data shipments between processes rather than large number of messages with smaller sizes. At implementation time, an implementor may decide to encode three attempts at a remote connection, rather than giving up after one attempt. At deployment time, a field agent can decide to use four replicas of a particular service rather than a single instance.

5.2 Scalability Analysis Technique

In this section we describe in detail the three main steps of our scalability analysis technique:

1. *Problem Identification:* we plot curves for the predicted QoS statistic from the performance model, as the scale factor is changed; next, we analyze the curves for potential scalability problems.
2. *Causal Analysis:* we determine possible causes for the scalability problem identified in step 1, based on our knowledge of the model and its sensitivity analysis.
3. *Problem Resolution:* we suggest design alternatives that can help remove or alleviate the scalability problem causes identified in step 2.

In each of these steps, the main challenge is to reason with a performance model of a *design*. An analysis of a design-time performance model has several possible sources of inaccuracies: (a) the model may be incomplete, as it does not model all aspects of the design at the same level of detail, (b) the parameters estimated for the model may be significantly different from the actual values in an implementation of the design, and (c) the stochastic models of the design behavior may introduce large variations in the QoS of interest, and yet we have to somehow reason with a representative statistic. We use *comparative and relative reasoning* to reduce the effects of such inaccuracies. We explain our reasoning techniques in the following.

For problem identification, we compare the QoS curves for representative transactions, and pick out transactions that perform poorly relative to other transactions. We require a minimum of two transactions for comparison. We then use Mean Value Analysis (MVA) techniques to understand if the problem identified by a comparison among transactions is a design issue or not. MVA allows us to make statements about the performance of a particular transaction with respect to the scale factor variations, based on standard queueing theory and component scalability assumptions. The main queueing theory assumption used is that no device gets overloaded (all devices have utilizations less than one), and we reason with mean values of metrics (e.g., response times) rather than a full distribution. The device utilization assumption is not limiting for our analyses: If any device utilization is greater than one, then using more of that device will improve the performance of the design. Therefore, it will not be beneficial to carry out further design-time scalability analysis for that design. The component scalability assumption we use are based on understandings of the design and standard computer science theory of algorithms (e.g., see [14]). For example, we make assumptions about how the response time of database search algorithms will scale with an increased load.

For causal analysis, we analyze the component response times for each model component. We then compare time spent in different components. Using this comparison and using results developed from a sensitivity analysis [1], we can identify the potential causes for a scalability problem. The sensitivity analysis isolates aspects of the design that are sensitive to the given scale factor. Hence, if a certain design component's response time is a large portion of the overall transaction response time, and if that design component's response time is sensitive to the scale factor, then it is likely to be a cause for the transaction's scaling problem.

For causal analysis and problem resolution, we compare the performance of different design alternatives and make statements about the relative scalability between the design alternatives. Design alternative comparisons are aided by the use of a scalability metric and statistical z tests, as described below.

Intuitively, for a scalability comparison of two designs A and B, we compare the QoS degradation as the scale factor varies. We can say that design A scales better than design B if the performance degradation observed for design A over a range of scale factor variations is less than the degradation for design B. We derive our scalability comparison metric, based on ideas from the ψ scalability metric defined by Jogalekar and Woodside [11] that compares the behavior of a design at two scale points 1 and 2.

Abstractly, we can view a system as a black-box that services various transactions. Operators issue new transactions to the system with a certain rate, λ (per second); the system processes the transactions and returns results to the operator after T seconds (response time). The ratio λ/T is called the power of the system [12], and it compares the rate of arrival of requests to the time it takes to service them. Jogalekar and Woodside have defined a scalability metric, ψ as follows:

$$\Psi = \frac{\frac{\lambda_1/T_1}{C_1}}{\frac{\lambda_2/T_2}{C_2}}$$

where

- λ_i is the rate of arrivals of customers at scale factor point *i*,
- T_i is the response time at scale factor point *i*, and
- C_i is the cost of the design deployment for scale factor point *i*

This metric compares the power of a system with regards to its cost, at different scale factor points. If scale factor points can be linearly ordered, and if point 2 is higher than point 1 in this order, then a lower value of ψ is more desirable. A ψ greater than 1 indicates poor scalability; all other values indicate good scalability. When comparing the scalability of design alternatives, the design alternative with a lower ψ indicates better scalability. We will illustrate the use of this metric for comparing the scalability of design alternatives in section 6.

For performance comparison of the design alternatives, we use statistical techniques to compare the significance of the differences in QoS from the two designs estimated by the performance model: if we are claiming a better response time for design A versus design B, we determine the confidence level for the mean estimated response time (MERT) of design A being lower than the MERT for design B. This analysis relies on the central limit theorem which allows us to assume that the MERTs have a normal distribution.

6 Scalability Analysis of Consul using Damson

In the scalability analysis of Consul, we instantiate our scalability concepts as follows. We study the effects of the scalability factor while using one operator, with one each of the several Consul services, all running on one workstation. The operator issues one of the three transactions, T1, T2, or T3 described in section 4.1. We have selected a minimal configuration for our analysis, and leave analysis of other configurations for future work.

Scale factor: *the number of managed devices*.

Scalability parameters:

- *Service times:* service times for individual Consul services increase logarithmically as a function of the number of managed devices.
- *Process Sizes:* Consul process sizes increase linearly, assuming 500 bytes per managed device.
- *Number of object processes:* Assume 10,000 objects within each object process. Since there will be at least one object per device, we add an object process per 10,000 devices.
- *Operator requests:* Initially the mean inter-arrival time for operator requests is set to five minutes. Mean inter-arrival time for operator request *decreases* by `numberofmanagedobjects`/100 ms as the scale factor changes.
- *Max Located Objects and Max Submap symbols:* These parameters give upper bounds for the number of managed objects that are accessed by individual transaction. The number of objects accessed by a particular transaction is given by a uniformly distributed random variable. We assume the upper bounds are 0.5% of the number of managed devices.
- *Environment service requests:* As the number of managed devices increase, we anticipate an increase in the transaction rate for transactions other than T1, T2, and T3. For example, these other transactions could update device information based on asynchronous device traps. Suppose that every 10 minutes a device generates a trap or event that needs an update in Consul. This implies that the inter-arrival rate for such events is $1.66\text{x}10^{-6}$ per ms per device. If we assume a service demand of 2ms for handling such demands, then we get a utilization of the CPU, per device of $1.67\text{x}10^{-6}\text{x}2 = 3.3 \text{ x } 10^{-6}$. Therefore, we make the overhead of the CPU device $(3.3\text{x}10^{-6})$ x `noOfManagedObjects`. Note that this assumption implies that

the environmental service-requests will saturate the CPU at about 300,000 devices.

Therefore, we restrict our study of this single operator case, to 100,000 managed devices[1].

S**calability tolerance:** Consul performance should not degrade substantially as the number of managed devices increase up to a million devices. Note that this requirement is defined intuitively in customer terms, where the meaning of “substantially” is subject to interpretation. In the following analysis, we do not rely on any particular interpretation of this term. We achieve this by comparing design alternatives rather than making absolute statements about any particular design.

Scalability limits: We want to evaluate the scalability of Consul as it manages small environments (hundreds of devices) and large environments (up to a million devices). The designers of Consul were also concerned about cost-of-ownership and use of Consul—therefore, we want to minimize the number of operators required to manage a given network.

We now present the problem identification, causal analysis, and problem resolution for Consul.

6.1 Problem Identification

In Figure 4 we present a graph of how the MERTs for the three transactions vary with the number of managed devices. As can be seen from the figure, T1 and T3 show a different behavior than T2. Whereas the T2 transaction’s MERT graph remains relatively flat over the duration of the scale factor variation, the graph for the MERT of the T1 transaction shows about a 13x increase. So, relatively speaking, there is a difference in how these three transactions are scaling. This causes us to probe further. The key question that we ask is: based on the structure of the design, what can we say about the response time variations for the T1 and T3 transactions, as we increase the number of managed devices? Is the behavior shown in the graphs intrinsic to the design or not? In particular, we want to know if there is any possibility of these response times decreasing as we increase the number of managed devices. The following claim, based on MVA techniques, answers this question. Before we make the claim, however, we first make an assumption that will be used in proving the claim.

Assumption 1 (Scale Assumption): As the number of devices managed by Consul increase,

- the CPU and disk service demands made by Consul components will increase or remain the same
- the operator transaction arrival rates for various classes of transactions (e.g., T1, T2, or T3) will increase or remain the same.

Claim 1: *According to the Consul model embedded in Damson, the mean response time for T1 and T3 transactions will not decrease with an increase in number of managed devices, regardless of errors in estimates for computation times.*

Proof: Although our design-time estimates for the various service demands may be off for various components of Consul modeled in Damson, we can use MVA arguments to establish this claim based on the scalability assumption presented above. Intuitively, our proof works as follows. The overall response time for a transaction is determined as the sum of residence times for the

1. This limit is for the configurations of Consul chosen for this experiment. With more hosts, this limit is likely to change.

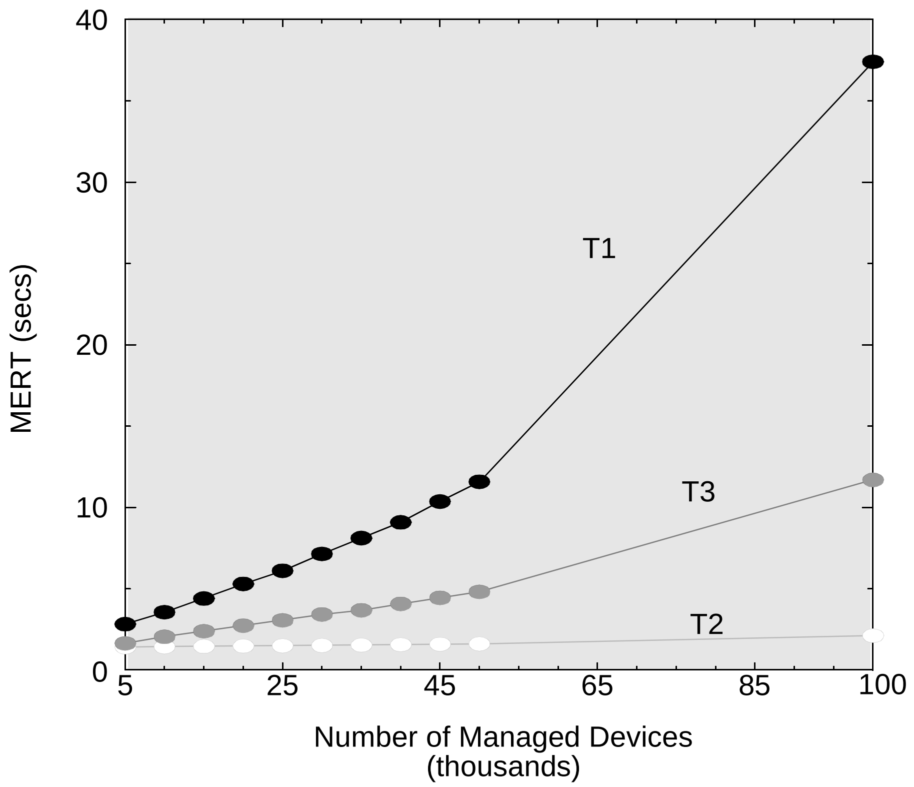

Figure 4. Estimated mean response times for T1, T2 and T3 Transactions, using Damson, as the number of managed devices increase. All Consul processes are on one host.

transaction at various queueing points (disk, CPU, or software locks). We show that the residence time at a particular queueing point can decrease only if either the service demand for that device decreases, or if the overall system transactions' arrival rates decrease. In particular, we show that the mean response time does not depend on the transaction arrival rate at a particular device; it only depends on the overall transaction's arrival rate. By our scalability assumption, however, we know that neither of these are expected to decrease as we increase the number of managed devices.

From MVA of open queueing network models with multiple classes of workloads [2], the following equation gives the mean response time for any class, c, of customers, $R_c(\vec{\lambda})$, for a particular input arrival rate vector, $\vec{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_C)$. In the equation, k is the number of devices in the model, and $R_{c,k}(\vec{\lambda})$ is the average residence time of customers class c at device k.

$$R_c(\vec{\lambda}) = \sum_{k=1}^{K} R_{c,k}(\vec{\lambda})$$

Note that for our purposes, a customer class is equivalent to a transaction type. For example, the T1, T2, and T3 transaction types each form a unique class of customers.

For delay servers, $R_{c,k}(\vec{\lambda})$ is $D_{c,k}$, the average service demand made by customers of class c on device k. For queueing servers, $R_{c,k}(\vec{\lambda})$ is given by the formula

$$R_{c,k}(\vec{\lambda}) = \frac{D_{c,k}}{1 - \sum_{j=1}^{C} U_{j,k}(\vec{\lambda})}$$

where $U_{j,k}(\vec{\lambda})$ is the average utilization of device k by a customer of class j.

In Damson, the queueing servers are: (1) CPU, (2) Disk, and (3) Software Locks. From Assumption 1, we know that the $D_{c,k}$ does not increase for the CPU and disk, as we increase the number of managed devices. Therefore, the only way we can decrease any $R_{c,k}(\vec{\lambda})$ is by either (1) decreasing $U_{j,k}(\vec{\lambda})$, or (2) decreasing the residence time at some software locks. We examine each of these cases below.

Case 1: For CPU and Disk utilizations, we know that

$$U_{j,k}(\vec{\lambda}) = \lambda_c D_{c,k}$$

where λ_C is the average throughput or arrival rate for customers of class c. We have already assumed with the scalability assumption that λ_C will not decrease; therefore the utilization cannot decrease.

Case 2: Software Locks: For software locks, we use a hierarchical model of queues. As shown in Figure 5, we model access to a critical section with a queue for obtaining a software lock, followed by a sub-network of queues for performing CPU and disk consumption during the critical section,

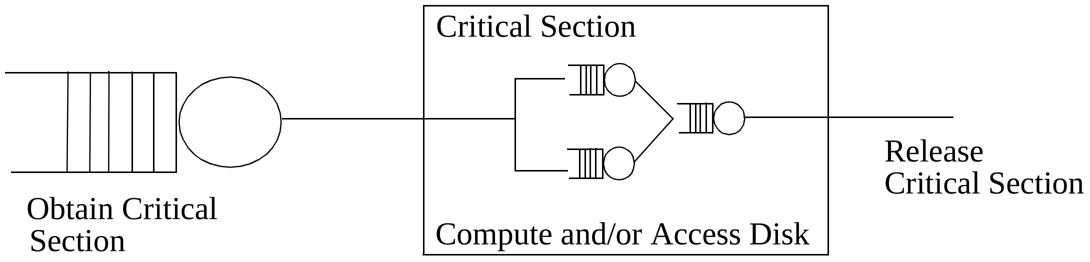

Figure 5. Queueing model of access to a critical section.

and followed by a release of the critical section lock. The overall residence time, R_{cs}, for a critical section (including the queueing for locks) is determined by the following formula:

$$R_{cs} = R_{sm}(1 + Q_{cs})$$

where R_{sm} is the residence in the critical section without the queue plus the service demand for obtaining a lock, and Q_{cs} is the expected queue length for obtaining the critical section lock. We have already established above that the residence time in the critical section component of R_{sm} will not decrease with the number of managed devices. Also, we know that the service demand for obtaining a lock is independent of the number of managed devices. Therefore, we now look at Q_{cs}, which can be expressed as

$$Q_{cs} = \frac{\frac{U_c(\vec{\lambda})}{C}}{1 - \sum_{j=1} U_j(\vec{\lambda})}$$

where U_c is the utilization of the critical section by a transaction of class C. We know that $U_c = \lambda_c D_c$. Therefore, Q_{cs} cannot decrease with an increase in managed devices, by our scalability assumption. Hence, claim 1.1 follows.

Since the `T1` transaction is most problematic (see Figure 4), we focus the rest of this study on the `T1` transaction.

Claim 2: *Adding hosts or disks will not significantly improve the mean performance of the T1 transaction of Consul for 100,000 managed devices.*

This claim is based on the fact that our simulation shows that the CPU, disk, and software locks utilization is at most 34% for 100,000 managed devices. The main increase in response time is due to the increasing service demand for the T1 transaction.

Based on Figure 4 and claims 1 and 2, our model and analysis predicts that there will be a scalability problem with the T1 transaction of Consul. To recapitulate, this prediction is based on the following facts: (1) the T1 transaction's MERT growth shows a sharp difference from the MERT growth of a parallel transaction, i.e., T2, (2) the structure of the design indicates that the response time of T1 can increase only as we increase the number of managed devices, it will not decrease, and (3) the device utilizations from the model simulations indicate that all devices are under-utilized—therefore, the response time of the transaction cannot be significantly impacted by adding hardware resources. This suggests that we must probe further into the design to see if we can make structural changes to the design and overcome this scalability limitation. We do this by performing a causal analysis on Damson as follows.

6.2 Causal Analysis

Damson enables a causal analysis through its ability to monitor the time spent in various model components. This analysis is performed by setting a switch in Damson enabling the generation of statistics, which can be imported into a spreadsheet for analysis. In our case, the estimated mean times spent in Damson components is shown in Figure 6.

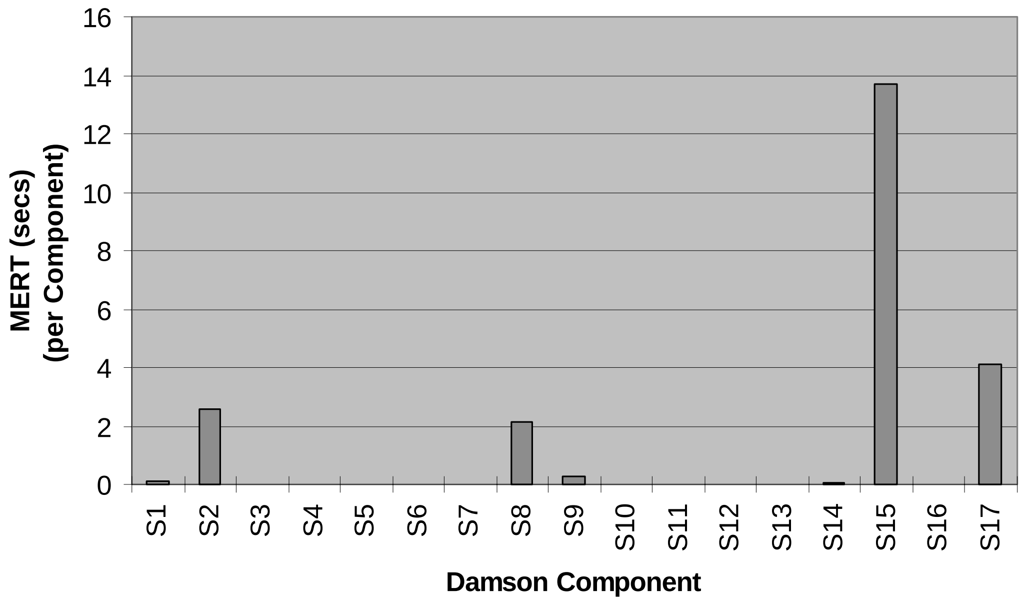

Figure 6. Time spent in various Damson Components (representing Consul Services), with 100,000 managed devices.

As can be seen from the Figure 6, most of the time is spent in service S15. Based on our design understanding, partially gained from the sensitivity analysis, we focus on two loops that call service S15 from S7 and S9. To test their effects, we short-circuit the two loops, i.e., we set the loop counts to 1. This leads to performance graphs as shown in Figure 7. The new performance

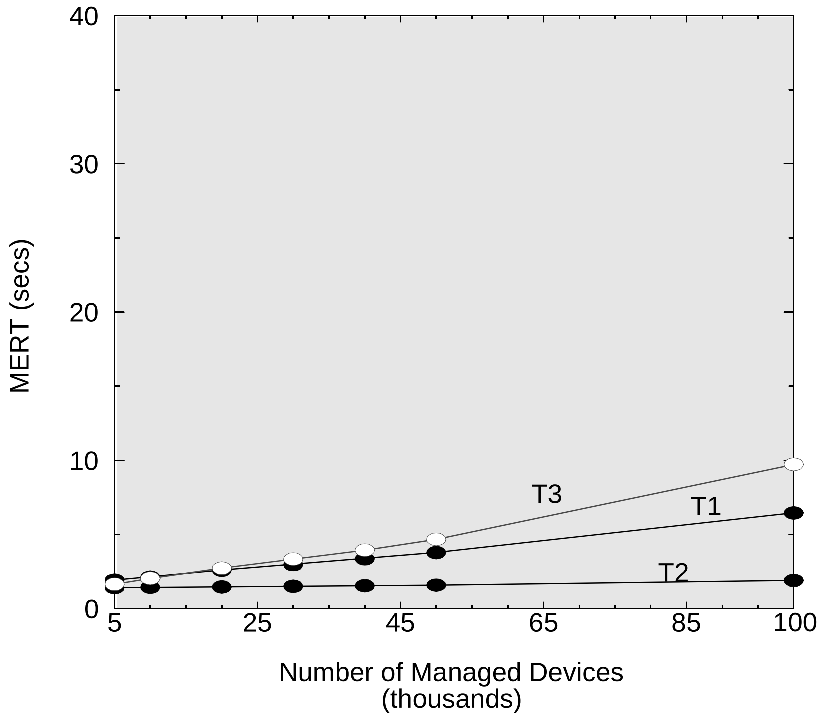

Figure 7. Graphs of transaction mean estimated response times when S7 and S9 make only one RPC call to service S15.

curves visually indicate that we can improve scalability of the design by removing the iterations. We can also compare the two growths *analytically*. The analytical comparison is based on the scalability metric discussed earlier.

Recall that the scalability metric compares the power to cost ratio for various scalability points. By comparing the power of the system at two scale points, the scalability metric allows for a degradation in performance as the workload increases. For Consul, the workload increase is represented by the scale factor, `noOfManagedObjects`. Since performance degradation is not desirable for Consul (even at higher workloads), we cannot use the power of the systems to compare the scalability of the design alternatives. Instead we compare response times with respect to the costs. Since we have already stated in Claim 2 that adding hosts or disks to the deployment of the design will not lower the response times, we do not consider cost changes for various scale factor points. Therefore, our scalability metric becomes

$$\Psi_i = \frac{T_i}{T_{Initial}}$$

for various scale points i. Figure 8 shows the Ψ-curves for the two designs: one with the loops and one without the loops.

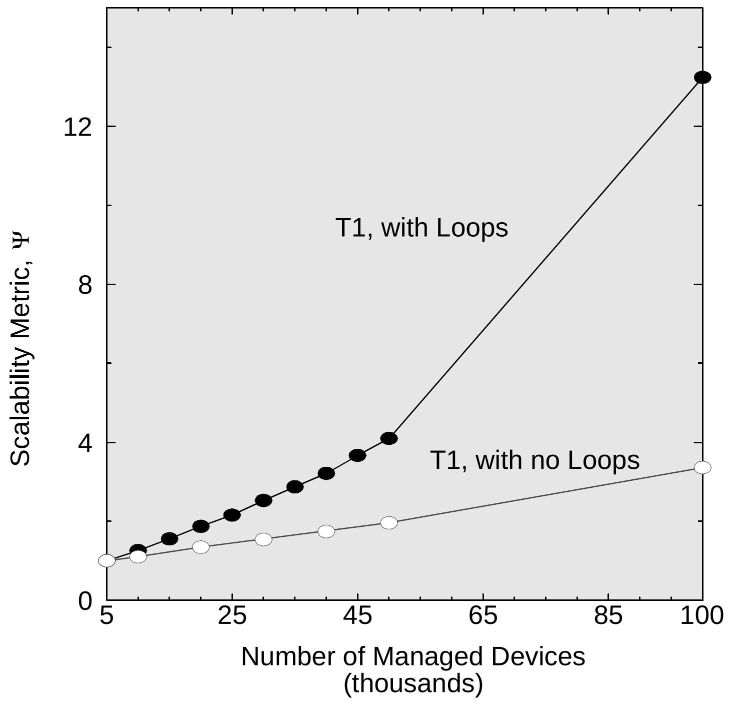

Figure 8. A comparison of the Scalability metric for the two design alternatives: (1) T1 transaction with loops between S7, S9 and S15, and (2) T1 transaction without the loops.

As can be seen from Figures 7 and 8, we have identified one primary cause for the performance and scalability difference for the T1 transaction—the **loop calls** to service S15. Based on our understanding of this cause, we can now state our second main result:

Claim 3: (Problem Analysis) *The iterative loops between S7 and S9 to service S15 are statistically significant performance problems for the T1 transaction of Consul.*

Proof: This claim is based on comparing the random variables that represent the mean response times for a Consul design with and without the two loops, represented by Figures 4 and 7, respectively. Let design X be the design of Consul with the loops, and design Y be the design of

Consul without the loops. Let $\{x_1, x_2, \ldots, x_m\}$ represent a sample of *m* estimated response times of the T1 transaction from the simulation of design X, and $\{y_1, y_2, \ldots, y_n\}$ be a sample of *n* estimated response times of the T1 transaction from the simulation of design Y. By the central limit theorem, we can assume that the means $\bar{x}$ and $\bar{y}$ have a normal distribution[1]. Hence, we can use the test statistic, z, to compare the two means [10]:

$$z = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{s_1^2}{m} + \frac{s_2^2}{n}}}$$

where s_1 is the observed standard deviation from simulation of design X and s_2 is the standard deviation from simulation of design Y. For a significance level $\alpha = 0.01$, $z_\alpha = 2.33$. Therefore, the MERT for design X will be higher than the MERT for design Y if $z > 2.33$. For the case under consideration, $\bar{x} = 34.88$, $s_1^2 = 642$, $m = 12$, $\bar{y} = 5.04$, $s_2^2 = 6.79$ and $n = 10$. From these numbers, we get $z = 4.05$. This clearly indicates that the estimated mean for design X is higher than the estimated mean for design Y, at a significance level of 0.01. For the z-test, we also know that the P-value is given by $P = 1 - \Phi(|z|)$. The P-value is the minimum significance level for the z-test to succeed. Since 4.05 is a pretty high z value (the tables do not list Φ), we only note that the P-value for $z = 3.49$ is 0.0002. Therefore, the P-value for our case is at least 0.0002, which for all practical purposes establishes our claim 3.

6.3 Problem Resolution

We now look at some scalability enablers, to get better scalability from the Consul design. At the design stage, one significant set of scalability enablers are potential design changes that can enable better performance at higher scalability points. Therefore, we look for design alternatives for Consul. Claims 1-3 that we have established so far, based on the sensitivity and scalability analyses, help focus our attention for design alternatives on the loops between S7, S9 and S15. We note that both these loops have the general pattern of a process A making queries to process B and associated objects for information. Depending on the design requirements, we can design such calls between processes A and B in one of three ways:

1. The current design, with synchronous RPC calls:

```
foreach query i {
    result = call process B;// RPC call
}
```

2. With asynchronous RPC’s:

```
foreach query i {
```

1. The values of x and y are obtained by running the simulations as a series of independent batches. The mean estimated response time for each batch is a value for x_i (or y_i). The batch size is arbitrarily selected based on a heuristic for a “long-enough” time interval (50Ksec). The number of batches is determined when the mean value for x (or y) coverges to a 5% confidence width with a 95% confidence level.

```
          CreateThread(...call process B...);
          // RPC call within a thread
     }
     wait for all children threads;
```

Design alternative 2 will require extra code to synchronize among the children threads to put together the result.

3. With bulk access:

```
     foreach query i {
          ComposeQuery(complexQuery, formQuery(i));
     }
     totalResult = call process B(complexQuery);
     // RPC call
```

where process A packages its queries into one message, makes the query to process B, and process B packages the query result in one message. This alternative assumes that such a packaging of queries is indeed possible, i.e., the n^{th} iteration computation in process A is independent of the results of the $(n - 1)^{th}$ iteration.

Since most of the cost in the response time of the `T1` transaction is due to service demands (as all the software and hardware resources are under-utilized), we do not expect design alternative 2 to reduce the response time very much. With design alternative 3, however, we can expect better response times as we will be reducing service demands on the ORB by having fewer calls, and saving the initial overhead for each call at process B. Therefore, we change Damson to model the third design alternative. We add a path through the ORB model to capture bulk messages. For bulk transfer operations, we multiply the mean number of bytes for the exponential distribution by a "bulk factor". (This bulk factor is set to the loop count random variable for the two services.) In the S7 loop for making ORB calls to S15 we change the loop iteration count to be 1 (instead of the original random variable), and we route the replies through the bulk ORB path, as we expect that the return result from the complex query be larger.

We plot $\Psi_{Design1}$ and $\Psi_{Design3}$ for the two designs in Figure 9. As can be seen from the plot, Ψ for design 3 is better than for design 1. The scalability metric for design 3 consistently outperforms the scalability metric for design 1. At the last point, for 100,000 managed devices, design 1 has a scalability metric a little more than twice that of design 3 (13.24 as compared to 6.2). Doing a similar analysis as we did for Claim 3, we have, $\bar{x} = 34.88$, $s_1^2 = 642$, $m = 12$, $\bar{y} = 13.48$, $s_2^2 = 57.23$ and $n = 10$. From these numbers, we get $z = 2.78$. This indicates that the MERT for the design with bulk accesses to the service S15 will be better than the individual, iterative accesses. This leads us to our final claim:

Claim 4: *Based on Damson, we predict that the response time for T1 transactions, design 3 will scale better than design 1.*

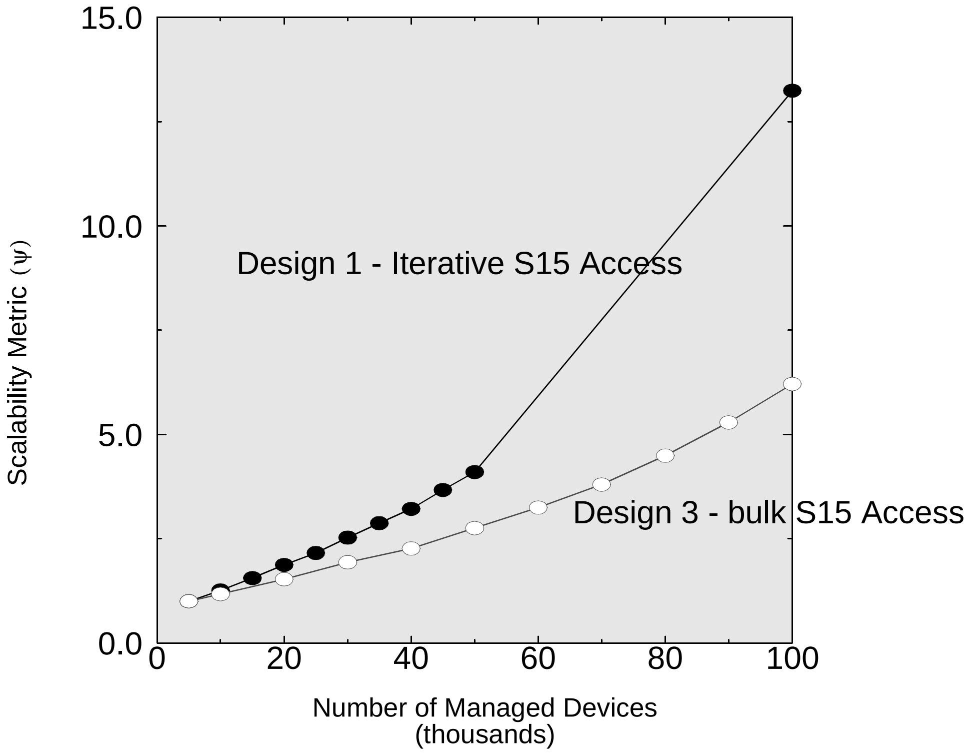

Figure 9. Scalability Metric plot for two alternative designs of Consul—(1) with an iterative access to the S15, and (2) with bulk access.

7 Conclusions

We have described a simulation model of Consul, a large-scale distributed network management system. We used the simulation model at design time to analyze the performance and scalability characteristics of Consul.

A major challenge for design time simulation is to derive conclusions that are valid, even in the presence of estimated rather than measured model input parameters. Parameter estimation, and thereby possible estimation errors, is an inherent property of design time modeling: we do not have access to an implementation that can be deployed and measured. We used scalability analysis to derive conclusions about Consul designs. The scalability analysis compares two designs, and determines which design is the more scalable. The conclusions drawn from the scalability analysis are stable with respect to estimation errors, because we show that errors will affect both designs in the same way, with respect to scalability properties.

In our study of Consul, we found that it is hard to determine the performance and scalability properties of large-scale distributed object systems using ad-hoc techniques. It may be possible to develop a sufficient ad-hoc understanding of individual components, but for a system with hundreds of asynchronous interacting components, an ad-hoc approach is simply not feasible. We found simulation to be an attractive approach for systematically analyzing performance and scalability. We found the main obstacle to be the level of abstraction and abstraction mechanisms offered by current simulation tools. Current tools for modeling computer systems are targeted primarily at hardware and networks. It was challenging in SES/Workbench to express application level concepts such as interacting asynchronous objects in a clean, high-level way. Moreover, the parameterization mechanisms for modeling tools does not offer sufficient support for describing design and configuration variations as model inputs. Current tools support only input in the form of parameter values; there is no notion of structural input such as the mapping of objects to physical resources.

Some lessons for future work are:

- With current software engineering practices, designers cannot justify the time to analyze models, but would rather use packaged analysis results. For this reason, we recommend the use of "analysis patterns," in similar vein as "design patterns" and "architectural patterns" are being used for object-oriented designs.
- We picked an off-the-shelf performance modeling tool, SES/Workbench, to develop our performance model of a CORBA-based application. The abstractions provided by the tool, however, were not sufficient for modeling the complexities of this application naturally, e.g., modeling process-host mapping and inter-process communication through the middleware. We recommend that a clean set of abstractions for modeling distributed applications be built that enable designers of distributed applications to develop such models themselves.
- The design-time analysis of a distributed applications' model is naturally based on several assumptions. Analysts need an arsenal of mathematical tools to present and defend their analysis and insights gained from the model and experimentation. We have identified early stages of such an arsenal, including parameter screening, design of experiments, scalability analysis, MVA, and concepts from probability and statistics. We recommend that these techniques be refined by applying them to another such large project, and subsequently tools be built to support their use.
- Dealing with a large number of parameters was difficult and frustrating for the design of experiments. It would have been better to have fewer parameters. Researchers at the University of Toronto and Carleton University in Canada have developed some useful abstractions for reducing the parameters for design-time models of distributed applications. We recommend a blend of the two approaches for accurate, abstract performance modeling of distributed applications.
- The model construction process can potentially go on indefinitely. We had to make sure that at some point we had a reasonable model to start analysis work. In general, this seems to be more of a property of the time available rather than any characteristic of the model itself.
- Some of the experiments took fairly long execution times, e.g. a few days of simulation. We could have used a distributed or parallel simulation environment for running our experiments.

- The work described here opens up several issues relating design-time analysis insights to deployment and operational management of distributed applications. For example, can the sensitivity analysis guide the instrumentation for operational monitoring; can a design-time model be converted into an embedded capacity planning model; can a P&S model be used to explore the space of possible deployment configurations?

8 Acknowledgments

Allan Shepherd made numerous contributions to this project and to earlier versions of this paper.

We thank the anonymous reviewers for their suggestions for improving the presentation and analysis.

We thank Thom Bartz, Trisha Johnson, Christina Mahon and Rick Whitner of the Hewlett-Packard Network and Systems Management Division for their participation with us in these studies and the numerous meetings, discussions, and suggestions. We thank Evan Kirhsenbaum, Joe Martinka, Keith Moore, Joe Sventek, and Tracy Sienknecht of the Software Technology Laboratory in Hewlett-Packard Laboratories for their suggestions for improving the analysis and its presentation. We thank Jerome Rolia, Prasad Jogalekar, Murray Woodside, and Fahim Sheikh of the Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, for their numerous discussions on various aspects of the analysis.

We thank Mary Loomis, Derek Coleman, and Tracy Sienknecht of the Hewlett-Packard Laboratories for their support and encouragement of this project.

9 References

[1] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley and Sons, Inc., 1991.

[2] E. Lazowska, J. Zahorjan, G. Graham, and K. Sevcik. *Quantitative System Performance*. Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, 1984.

[3] D. A. Menasce, V. A. F. Almeida, and L. W. Dowdy. *Capacity Planning and Performance Modeling*. Prentice Hall, Englewood Cliffs, NJ 07632, 1994.

[4] G. Hills, J. Rolia, and G. Serazzi. *Performance Engineering of Distributed Software Process Architectures*. In *Quantitative Evaluation of Computing and Communication Systems*, Lecture Notes in Computer Science, 977, pages 357–371. Springer-Verlag, Heidelberg, 1995.

[5] Scientific and Engineering Systems (SES), 4301 Westbank Drive, Bldg. A, Austin, TX 78746. *SES Workbench*, 1995.

[6] J. Rolia and K. Sevcik. The Method of Layers. *IEEE Transactions on Software Engineering*, 21(8):689–700, August 1995.

[7] F. Sheikh, J. Rolia, P. Garg, S. Frolund, and A. Shepherd. Layered Performance Modeling of a CORBA-based Distributed Application Design. *Proceedings of the 1st World Congress on Com-*

puter Simulation, Singapore, September 1997.

[8] N. M. Muller. *Focus on OpenView: A Guide to Hewlett-Packard's Network and Systems Management Platform*. CBM Books, Fort Washington, PA, 1995.

[9] J. P. C. Kleijnen, Sensitivity Analysis and Optimization in Simulation: Design of Experiments and Case Studies. *Proc. of the 1995 Winter Simulation Conference*, pages 133-140, December 1995.

[10] J. L. Devore, *Probability and Statistics for Engineering and the Sciences*, Brooks/Cole Publishing Company, Monterey, Calif., 1987.

[11] P. P. Jogalekar and C. M. Woodside, A Scalability Metric for Distributed Computing Applications in Telecommunications, Sept., 1996, *Submitted for publication*.

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

[13] V. Fernandes, J. C. Browne, D. Neuse, R. Velpuri, Some Performance Models of Distributed Systems, Proceedings of the Computer Measurement Group (CMG) Conference 1986, pp. 30-37.

[14] D. Harel, *Algorithmics: The Spirit of Computing*, Addison-Wesley Publishing Company, Wokingham, England, 1987.

[15] K. M. Chandy, J. Mishra, R. Berry and D. Neuse, "The Use of Performance Models in Systematic Design," *AFIPS Fall Joint Computer Conference*, Vol. 51, pp. 251-256, 1982.

[16] B. D. Noble and M. Satyanarayanan, "An Empirical Study of a Highly Available File System," *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 16-20, 1994, pp. 138-149.

[17] E. D. Lazowska, J. Zahorjan, D. R. Cheriton and W. Zwaenepoel, "File Access Performance of Diskless Workstations," *ACM Transactions on Computer Systems*, Vol. 4, No. 3, August 1986, pp. 238-268.

[18] C. A. Thekkath, J. Wilkes and E. D. Lazowska, "Techniques for File System Simulation," *Software-Practice and Experience*, Vol. 24(11), 981-999 (November 1994).

[19] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham and M. J. West, "Scale and Performance in a Distributed File System," *ACM Transactions on Computer Systems*, Vol. 6, No. 1, February 1988, pp. 51-81.

[20] A. Delis and N. Roussopoulos, "Performance Comparisons of Three Modern DBMS Architectures," *IEEE Transactions on Software Engineering*, Vol. 19, No. 2, February 1993, pp. 120-138.

[21] M. Carey, M. Franklin, M. Livny and E. Sheikita, "Data Caching Tradeoffs in Client-Server DBMS Architectures," In *Proceedings of the 1991 ACM SIGMOD International Conference*, Denver CO, May 1991.

[22] J. E. Neilson, C. M. Woodside, D. C. Petriu and S. Majumdar, “Software Bottlenecking in Client-Server Systems and Rendezvous Networks,” *IEEE Transactions on Software Engineering*, vol. 21, no. 9, pp. 776-782, September 1995.

[23] C. U. Smith, “Independent General Principles for Constructing Responsive Software Systems,” *ACM Transactions on Computer Systems*, Vol. 4, No. 1, (Feb. 1986), pp. 1-31.

[24] C. U. Smith, *Performance Engineering of Software Systems*, Addison-Wesley, 1990.

[25] O. C. Ibe H. Choi and K. S. Trivedi, “Performance Evaluation of Client-Server Systems,” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 11, November 1993, pp. 1217-1229.