*H. Gellersen et al. (eds.), ECSCW 2005: Proceedings of the Ninth European Conference on Computer-Supported Cooperative Work, 18-22 September 2005, Paris, France*, 83–102.
© 2005 *Springer. Printed in the Netherlands.*

# Formally Analyzing Two-User Centralized and Replicated Architectures

Sasa Junuzovic, Goopeel Chung* & Prasun Dewan
Department of Computer Science
University of North Carolina at Chapel Hill, NC, USA
Department of Computer and Information Science
Westfield State College, MA, USA*
sasa@cs.unc.edu, gchung@cs.wsc.ma.edu, dewan@cs.unc.edu

**Abstract.** We have developed a formal performance model for centralized and replicated architectures involving two users, giving equations for response, feedthrough, and task completion times. The model explains previous empirical results by showing that (a) low network latency favors the centralized architecture and (b) asymmetric processing powers favor the centralized architecture. In addition, it makes several new predictions, showing that under certain practical conditions, (a) centralizing the application on the slower machine may be the optimal solution, (b) centralizing the application on the faster machine is sometimes better than replicating, and (c) as the duration of the collaboration increases, the difference in performances of centralized and replicated architectures gets magnified. We have verified these predictions through new experiments for which we created synthesized logs based on parameters gathered from actual collaboration logs. Our results increase the understanding of centralized and replicated architectures and can be used by (a) users of adaptive systems to decide when to perform architecture changes, (b) users who have a choice of systems with different architectures to choose the system most suited for a particular collaboration mode (defined by the values of the collaboration parameters), and (c) users locked into a specific architecture to decide how to change the hardware and other collaboration parameters to improve performance.

## Introduction

Two main architectures have been used to support the sharing of a program among multiple users: centralized and replicated. In the centralized architecture, the shared program executes on a computer belonging to one of the collaborators, receiving input from and broadcasting output to all users. In the replicated architecture, a separate replica of the program executes on the computer of each user, receiving input from all users and producing output for only the local user.

Both architectures have been popular in commercial and research systems. In fact, often a single collaborative system supports both architectures. For example, NetMeeting and Webex provide the centralized architecture for application sharing and the replicated architecture for whiteboard sharing. Chung and Dewan (2001) allow the choice of architecture to be made at application start time while their later results (2004) allow it to change at runtime.

The choice of the architecture affects the semantics, correctness, and performance of the shared program. In this paper, we focus on performance. Previous studies on the performance of collaborative architectures have been restricted to gathering empirical data. To the best of our knowledge, no previous work has developed an analytical performance model.

An analytical model is an attractive idea for two main reasons. First, like analytical models in other computer science fields, it increases our understanding of the subject analyzed. In the case of collaboration architectures, it helps us better understand and compare the event flow and performance of the centralized and replicated architectures. Second, empirical data can inform us about the performance of a collaborative application only under the collaboration conditions used in the measurements. In general, there exists an infinite design space of collaboration modes defined by a variety of collaboration parameters such as network latency and processing power. An analytical model can predict the performance for the entire design space modeled.

As a first step towards meeting these two goals, we have developed a formal performance model for centralized and replicated architectures involving two users, giving equations for response, feedthrough, and task completion times. Our model takes into account collaboration parameters such as network latency, processing powers of the computers used, command-processing time, think time, and degree of participation of each user in the collaboration. The model provides a better understanding of the event flow in the centralized and replicated architectures. It also explains previous empirical results and makes several new predictions. We have verified these predictions through new experiments for which we synthesized logs based on parameters gathered from actual collaboration logs.

The rest of this paper is organized as follows. We first present related work on the performance of collaborative architectures. Next, we develop the mathematical equations comprising the analytical model. We then validate our model against empirical data shown in previous work and through new experiments. Finally, we end with conclusions and directions for future work.

## Related Work

Unlike in traditional computer science fields such as databases and operating systems, there has been relatively little work in the collaboration domain on studying the performance of system architectures, even though, arguably, performance is more important in this field because of the human in the event-processing loop. As mentioned earlier, existing studies have been confined to gathering empirical data. Moreover, there have been very few studies that have directly targeted collaboration. One can, however, make some collaboration implications indirectly from studies of distributed window systems.

Nieh, Yang, Novik et al. (2000) conducted experiments that measured the relative performances of two distributed window systems, the Linux implementation of VNC (Hopper, 1998) and Microsoft's Windows 2000 RDP implementation. The architecture used was essentially a two-user centralized architecture with the user at the hosting site inactive. Such a setup gives an idea of the performance experienced by a remote user interacting with a centralized program, assuming the host site does not become a bottleneck. These studies compared two different implementations of the centralized architecture and do not addresses the relative performances of different architecture configurations.

Wong and Seltzer (2000) measured the network load for various remote user operations. Danskin and Hanrahan (1994) measured the frequencies of these operations. Together, these two results give an idea of the actual bandwidth requirements for a variety of remote desktop tasks. Two other studies, one involving Microsoft's Terminal Services (NEC, 2000) and the other by Droms and Dyksen (1990) showed that average and maximum network bandwidth requirements of remote desktop operations can vary greatly. Again, these studies do not address the relative performances of different architecture configurations. This limitation was addressed by the following two works.

Ahuja, Ensor, Lucco et al. (1990) performed experiments to compare the network load imposed by the centralized and replicated implementations of a shared drawing program. They found the following three results. (1) When output was not buffered, there were 6 times as many output events as input events. (2) When output was buffered, there were 3.6 times as many output events as input events. (3) An output event was about the same size as an input event – about 25 bytes (presumably not including network headers).

Like Ahuja, Ensor, Lucco et al. (1990), Chung and Dewan (2004) compared the centralized and replicated architectures, addressing response and task completion times instead of network load. They showed that (a) low network latency favors the centralized architecture and (b) asymmetric processing powers favor the centralized architecture. As these conditions can change dynamically, they developed a system that supports architecture changes at runtime. They also performed experiments showing that when a user with a powerful computer joins the collaboration, it is useful to dynamically centralize the shared program to the new user's computer.

We use previous work to identify collaboration parameters relevant to performance, and we extend it by (a) defining an analytical model that explains existing results about response, feedthrough, and task completion times, and (b) performing new kinds of experiments that validate results predicted by the analytical model not shown previously.

## Formal Analysis

As mentioned above, we use response, feedthrough, and task completion times as performance metrics. We define the response (feedthrough) time of a command to be the time that elapses from the moment the command is input to the moment the inputting (non-inputting) user views the corresponding output. We define the task completion time for a particular user as the time that elapses from the moment the collaboration session begins to the moment the user sees the final output. We have not yet considered other important metrics such as jitter (Dyck and Gutwin, 2004).

Developing an analytical model is a complex task, especially when deriving the task completion time for the replicated architecture. Therefore, we make certain assumptions in this first cut at a performance model of collaboration architectures. The major assumption we make is that the collaboration involves only two users, who we denote as $user_1$ and $user_2$. We will describe other assumptions as we introduce our collaboration parameters:

*Processing powers of collaborators' computers*: As shown by earlier work, a centralized architecture may offer better performance than a replicated architecture when the difference between the processing powers of the users' computers is high as the faster computer can act as a high-performance server for compute-intensive tasks. Thus, it is important to consider processing powers in our equations. We assume that $user_1$'s and $user_2$'s machines have processing powers of $p_1$ and $p_2$ MHz, respectively. We assume that the work required, measured in CPU cycles, is the same for all input commands and refer to it as $w$. Without loss of generality, we assume that $p_1 > p_2$, that is, $user_1$ has a faster computer than $user_2$. Thus, $w/p_1 < w/p_2$, or in other words, an input command is processed faster on $user_1$'s computer than on $user_2$'s computer.

*Network latency*: Previous work has also shown the influence of network latency on response and task completion times. In a centralized architecture, network latency affects the feedback time of the remote user. In both architectures, it influences both users' feedthrough times, and hence the task completion time when there is coupling between the two users, that is, when a user cannot input the next command before he sees the output for the previous command entered by the other user. For simplicity, we assume that the network latency between the two machines is constant and denote it as $d$.

*Number of input commands by each user*: This can have a large impact on the performance of an architecture. To illustrate, assume a centralized architecture in which one of the users provides all the input. If the active user's computer is hosting the program, then the task completion time is independent of network delays. This is not the case if the active user is on the other computer. We will see later that not only is the degree of participation of each user important but also the total of input commands by each user, as differences in the performances of architectures can get magnified the longer the collaboration lasts. We let $c_1$ and $c_2$ denote the total number of input commands by $user_1$ and $user_2$, respectively.

*Think time before each input command*: We assume that the think time for each input command is constant and denote it as $t$.

*Number of answered* $user_1$*'s input commands*: An input command by $user_1$ is answered if $user_2$ inputs a command after it. Thus, the number of $user_1$'s answered commands equals the number of input commands by $user_1$ unless $user_1$ inputs last. In this case, the last burst of input commands by $user_1$ is not answered. We let $a$ denote the number of $user_1$'s answered commands. Note that $a \leq c_1$.

*Number of input subsequences*: An input subsequence ends and a new one begins every time $user_2$'s input command is followed by $user_1$'s command. Also, an implicit start and end of the first and last subsequences occur at the start and end of the collaboration session. As we will see, during each subsequence in the replicated architecture case, there are periods of time during which $user_1$'s faster computer is waiting for $user_2$'s slower computer to catch up. These idling periods are important because they affect the replicated architecture's task completion time. We denote the number of subsequences by $s$.

*Number of user input commands in a subsequence*: We assume that $user_1$ is the first user to enter a command. Recall that $user_1$ has $c_1$ and $user_2$ has $c_2$ total input commands, and that both users input in each subsequence, except when $a < c_1$, that is, when only $user_1$ inputs in the last subsequence. We let $c_{1,i}$ and $c_{2,i}$ denote the number of input commands in subsequence $i$ by $user_1$ and $user_2$, respectively.

While we know of no data identifying the number of users in a computer-supported collaborative session, we believe a significant number of such sessions involve two users based on the fact that most telephone conversations are two-way calls, many research papers have two authors, many games involve two users, and pair programming requires exactly two users. Thus, our two-user assumption does not make our work impractical. Assuming constant command-processing times is reasonable in whiteboards, instant message clients, games, and several other widely used programs that offer a small number of commands of similar complexity. For example, a command to draw a rectangle is processed very similarly to one that draws an oval. Assuming constant low think times is reasonable in closely-coupled interactions in which it is considered impolite to keep collaborators waiting. When think times are large, assuming they are constant is less reasonable, but large think times dominate the task completion time, and as a result, their specific values do not matter as much. Even if some of these assumptions are considered somewhat simplistic, as mentioned above, the main goal of this paper is to motivate research in analytical performance models of collaboration architectures rather than be the last word on them.

In two-user collaborations there are three architectures to consider: (1) the shared program is centralized on the faster computer, (2) the program is centralized on the slower computer, and (3) the program is replicated. The reason for considering (2) is that in a centralized system, the user initiating the collaboration is the one whose computer hosts the program. By comparing (1) and (2) we can estimate the benefit of (a) adding a constraint on users that requires the one with the faster computer to initiate the session in a static system and (b) changing the architecture at runtime in a dynamic system. Below, we derive the response, feedthrough, and task completion times for these three architectures.

## Response Times in Centralized Architectures

In cases where a user's input commands are processed by processed by the local program, the local program replica processes each input command immediately after the local user provides it, keeps processing it without interruption, and finally generates an output message. Hence, if $user_1$ is hosting the centralized architecture, his response time is $w/p_1$, and if $user_2$ is hosting the centralized architecture, his response time is $w/p_2$. Now, consider the case where a user's input commands are processed by processed by a remote program instance. Each of his input commands must first travel to the remote program instance, which then immediately starts processing the command upon receipt. The remote program instance processes the input command without interruption, and finally generates an output message, which, then, must travel back to the input provider's computer. Therefore, $user_2$'s local response time is $2d+w/p_1$ in centralized architectures with $user_1$ hosting, and $user_1$'s local response time is $2d+w/p_2$ in centralized architectures with $user_2$ hosting. Hence, we have

| | | |
|---|---|---|
| $Resp_{Cent}U_1 = w/p_1$ | if $user_1$ is hosting | [Eq. 1.1] |
| $Resp_{Cent}U_1 = 2d+w/p_2$ | if $user_2$ is hosting | [Eq. 1.2] |
| $Resp_{Cent}U_2 = w/p_2$ | if $user_2$ is hosting | [Eq. 1.3] |
| $Resp_{Cent}U_2 = 2d+w/p_1$ | if $user_1$ is hosting | [Eq. 1.4] |

## Response Times in Replicated Architectures

In the replicated architecture, each user's input command is processed by the local replica without synchronizing with the other replica. Thus, $user_1$'s and $user_2$'s response times in the replicated case are $w/p_1$, and $w/p_2$, respectively. Hence,

$Resp_{Rep}U_1 = w/p_1$ [Eq. 2.1]

$Resp_{Rep}U_2 = w/p_2$ [Eq. 2.2]

## Task Completion Times in Centralized Architectures

We calculate here the centralized architecture task completion time for *$user_1$* and assume *$user_1$* inputs first. Figure 1(a) below illustrates the elements of an example task completion time in a centralized architecture scenario where the shared program is centralized on *$user_1$*'s faster computer. In this interaction sequence, *$user_1$* enters inputs 1, 2, 5, and 6, while *$user_2$* enters inputs 3, 4, and 7. Let $L_i$ denote the $i$th contiguous period of time during which the centralized program is busy *locally* processing or waiting for input commands from the *local* user, *$user_1$.* Let $R_i$ denote the $i$th contiguous period of time during which the program is waiting for input commands from the *remote* user, *$user_2$*. Since a collaboration session starts when the centralized program starts processing the first input command of the session and ends when it finishes processing the last input command of the session, $L_i$ must immediately be followed by $R_i$, which then must be immediately followed by $L_{i+1}$. Therefore, the total task completion time equals $\Sigma L_i + \Sigma R_i$.

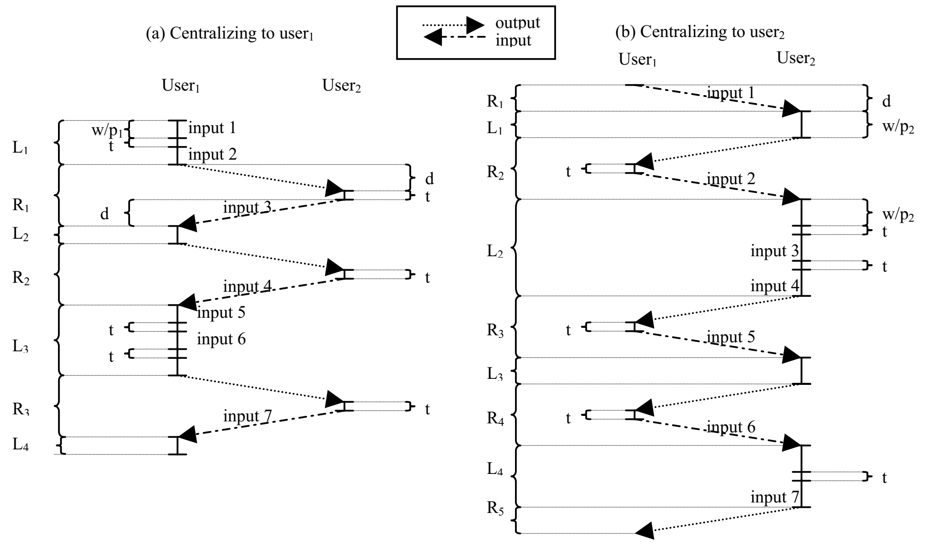

Figure 1. Centralized Architecture Time Diagram

$\Sigma L_i$: This term includes two components: the time used to *locally* process the input commands of both users and the think times of the *local* user, *$user_1$*. The first component is $(c_1+c_2)w/p_1$ and the second one is $(c_1-1)t$ as *$user_1$* spends $t$ think time before inputting a command for all commands but the first. Thus, we have

$$\Sigma L_i = (c_1+c_2)w/p_1+(c_1-1)t \qquad \text{[Eq. 3]}$$

$\Sigma R_i$: As shown in Figure 1(a), waiting for an input command from *$user_2$* consists of three parts: the network delay in transmitting the output for the previous command from *$user_1$*'s computer to *$user_2$*'s computer, the think time $t$ of

*$user_2$* before inputting the next command, and finally the network delay in transmitting *$user_2$*'s input command to *$user_1$*'s machine. Thus, we have

$\sum R_i = c_2(2d+t)$ [Eq. 4]

From equations 3 and 4, we can derive the total task completion time for a centralized architecture in which the faster user, *$user_1$*, is the host.

$task_{cent}U_1 = \sum L_i + \sum R_i = (c_1+c_2)w/p_1+(c_1+c_2-1)t+2c_2d$ [Eq. 5.1]

As the three terms above show, it consists of the time required to process all input commands, the time it takes to think before all commands but the first one, and the network delays incurred in receiving the input commands from and in sending the outputs of the previous commands to the remote user.

If we consider Figure 1(b), then we can similarly reason about the task completion time of a centralized architecture in which the slower user, *$user_2$*, is the host. In this case, all the processing is done by *$user_2$*'s computer, and the delays are incurred for *$user_1$*'s commands. As this the dual of the previous case, the task completion time mirrors equation 5.1.

$task_{cent}U_2 = \sum L_i + \sum R_i = (c_1+c_2)w/p_2+(c_1+c_2-1)t+2c_1d$ [Eq. 5.2]

## Task Completion Time in Replicated Architecture

Deriving the replicated architecture task completion time is significantly different and more complicated than deriving it for the centralized case for several reasons. First, the faster computer may have to wait for the slower one to catch up because of processing time differences. Second, this wait occurs, not after each input command, but instead, when control switches from the user with the fast computer to the user with the slow computer. Finally, the wait time depends not only on the processing power difference but also on the network delays and think times.

Figure 2 below illustrates the elements of an example task completion time in the replicated architecture scenario during which *$user_1$* enters inputs 1, 2, 5, and 6, and *$user_2$* enters inputs 3, 4, and 7. This example illustrates our derivation of the task completion time. We will calculate the task completion time for *$user_1$* only. As before, let $L_i$ denote the $i^{th}$ contiguous period of time during which the program on *$user_1$*'s computer is busy *locally* processing input commands or waiting for input from the *local* user, *$user_1$*. Let $R_i$ denote the $i^{th}$ contiguous period of time during which the program on *$user_1$*'s computer is waiting for input commands from the *remote* user, *$user_2$*.

*$\sum L_i$*: As in the centralized case, the shared program on *$user_1$*'s computer must process all of the input commands and wait for the think time, *t*, before each command entered by *$user_1$* except the first one. Thus, we have

$\sum L_i = (c_1+c_2)w/p_1+(c_1-1)t$ [Eq. 6]

*$\sum R_i$*: In order to calculate the time the faster computer waits for the slower one, we divide the task completion time into subsequences and then add up the

time all the subsequences contribute. A subsequence *i* consists of $c_{1,i}$ *user₁*'s consecutive input commands followed by $c_{2,i}$ *user₂*'s consecutive input commands. In case *user₁* provides the last input command in a subsequence, the last subsequence is composed only of *user₁*'s input commands. We refer to such a subsequence as a half subsequence as opposed to a full subsequence. Therefore, a task sequence is composed of full subsequences and possibly another one half subsequence. The first subsequence is different from the others in that both computers are ready to process the first input command in the subsequence. Therefore, we treat it differently from the others. We now calculate $\sum R_i$ in terms of its components.

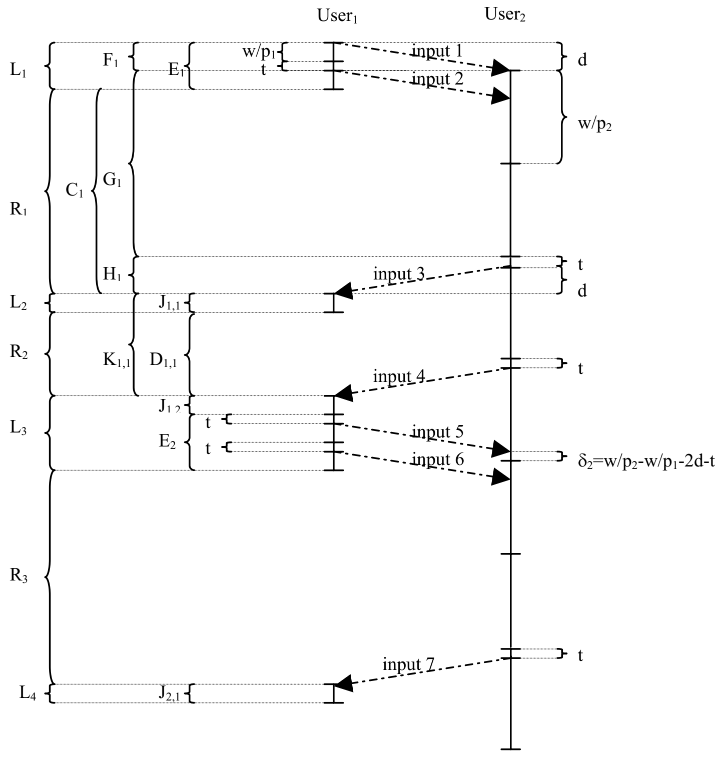

Figure 2. Replicated Architecture Time Diagram

$\sum R_i$ in First Subsequence

*$C_1$*: $C_1$ is defined as the time that elapses from the moment *user₁*'s program replica finishes processing the last input command by *user₁* in the subsequence to the moment it begins processing the first input command by *user₂* in the subsequence. Figure 2 graphically shows that $C_1+E_1=F_1+G_1+H_1$. Therefore, $C_1=(F_1+G_1+H_1)-E_1$. We next calculate the values on which $C_1$ depends.

*$E_1$*: $E_1$ is defined as the time that elapses from the moment *user₁*'s program replica begins processing *user₁*'s first input command in the subsequence to the

moment it finishes processing *user₁*'s last input command in the subsequence. This includes processing $c_{1,1}$ input commands by $user_1$ and a think time of $t$ for each of these commands except the first. Thus, we have

$E_1 = c_{1,1}w/p_1+(c_{1,1}-1)t$ [Eq. 7]

$F_1$: $F_1$ is defined as the time that elapses from the moment $user_1$'s first input command in the subsequence leaves $user_1$'s computer to the moment $user_2$'s program replica begins to process it. Therefore, $F_1$ is the network delay between the users' computers, $d$. Thus, we have

$F_1 = d$ [Eq. 8]

$G_1$: $G_1$ is defined as the time that elapses from the moment $user_2$'s program replica begins processing $user_1$'s first input command in the subsequence to the moment it finishes processing $user_1$'s last input command in the subsequence. There are two cases to consider here based on whether the think time, $t$, is less than the difference in processing times of an input, $w/p_2-w/p_1$. If $t \leq w/p_2-w/p_1$, then $user_2$'s computer will never be idle waiting for a command to arrive from $user_1$'s computer (Figure 2), except initially. In this case, $G_1 = c_{1,1}w/p_2$, the time required to process $user_1$'s input commands in the subsequence on $user_2$'s computer. But if $t > w/p_2-w/p_1$, $\delta_1 = t-(w/p_2-w/p_1)$, $user_2$'s program replica will finish processing $user_1$'s previous input command by the time it receives $user_1$'s next input command. Thus, $G_1$ increases by $\delta_1(c_{1,1}-1)$, which is the time $user_2$'s computer is idle while $user_1$' inputs in the first subsequence (Figure 3). Hence,

$G_1 = c_{1,1}w/p_2$ if $\delta_1 = t-(w/p_2-w/p_1) \leq 0$ [Eq. 9.1]
$G_1 = c_{1,1}w/p_2+\delta_1(c_{1,1}-1)$ if $\delta_1 = t-(w/p_2-w/p_1) > 0$ [Eq. 9.2]

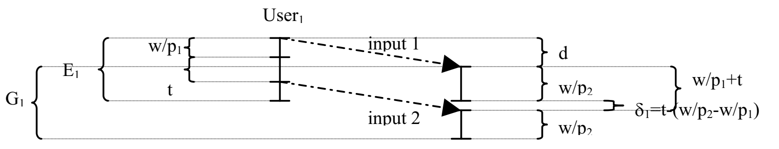

Figure 3. Illustrating $G_1$ if $t > w/p_2-w/p_1$

$H_1$: $H_1$ is defined as the time that elapses from the moment $user_2$'s program replica finishes processing $user_1$'s input commands in the subsequence to the moment $user_1$'s program replica begins to process $user_2$'s first input command in the subsequence. Once $user_2$'s program replica finishes processing $user_1$'s commands in the subsequence, $user_2$ spends $t$ time thinking about the output of $user_1$'s last input command and then enters his first input command in the subsequence. Therefore, $H_1$ consists of $user_2$'s think time, $t$, and the network delay between the users' computers, $d$. Thus, we have

$H_1 = t+d$ [Eq. 10]

$user_1$'s replica begins processing $user_2$'s first input command in the subsequence immediately. Therefore, by the definitions of $C_1$, $E_1$, $F_1$, $G_1$, and $H_1$, it must be the case that $C_1 = (F_1+G_1+H_1)-E_1$. Based on equations 7, 8, 9.1, 9.2, and 10, the wait time that elapses from the moment *user$_1$*'s replica finishes processing *user$_1$*'s last input command in the subsequence to the moment it begins processing *user$_2$*'s first input command in the subsequence, is

$C_1 = 2d+c_{1,1}(w/p_2-w/p_1)-(c_{1,1}-2)t$ if $t-(w/p_2-w/p_1) \leq 0$ [Eq. 11.1]
$C_1 = 2d+(w/p_2-w/p_1)+t$ if $t-(w/p_2-w/p_1) > 0$ [Eq. 11.2]

The $C_1$ component of $\sum R_i$ in the first subsequence tells us how long *user$_1$*'s faster computer must wait for *user$_2$*'s first input command in the subsequence after processing all of *user$_1$*'s input commands in the first subsequence. However, $\sum R_i$ in the first subsequence also includes the time *user$_1$*'s computer must wait for *user$_2$*'s computer while processing *user$_2$*'s input commands in the first subsequence. *user$_1$*'s program replica will wait from the moment it processes *user$_2$*'s $j^{th}$ command in the subsequence until *user$_2$*'s $j$+1st command of the subsequence arrives. This time is equal to $D_{1,j}$ and for *user$_2$*'s $j^{th}$ input command. Figure 2 shows $D_{1,1}$.

$\sum D_{1,j}$: $\sum D_{1,j}$ is the summation of the wait times during the time period in which *user$_1$*'s program replica is processing *user$_2$*'s input commands in the first subsequence. We now show that $\sum D_{1,j}$ equals $\sum (K_{1,j}-J_{1,j})$.

$K_{1,j}$: $K_{1,j}$ is defined as the time that elapses from the moment *user$_1$*'s program replica begins processing *user$_2$*'s $j^{th}$ command in the subsequence until *user$_1$*'s program replica begins processing *user$_2$*'s $j$+1st command in the same subsequence. Since $p_1 > p_2$ and $t \geq 0$, we have $w/p_1 < t+w/p_2$, that is, the time *user$_1$*'s program replica takes to process *user$_2$*'s input command, $w/p_1$, is less than the time it takes *user$_2$*'s program replica to process the same input command, $w/p_2$, and the time, $t$, during which *user$_2$* thinks before inputting his next command. As a result,

$K_{1,j}= w/p_2+t$ [Eq. 12]

$J_{1,j}$: $J_{1,j}$ is the time that *user$_1$*'s replica requires to process *user$_2$*'s input command. Hence,

$J_{1,j}= w/p_1$ [Eq. 13]

By the definition of $D_{1,j}$, $K_{1,j}$, and $J_{1,j}$, $D_{1,j}= K_{1,j}-J_{1,j}$. As a result, from equations 12 and 13, we have

$\sum D_{1,j} = \sum (K_{1,j}-J_{1,j}) = (c_{2,1}-1)(t+w/p_2-w/p_1)$ [Eq. 14]

In other words, when *user$_2$* is inputting a command, *user$_1$*'s computer must wait for the think time and extra time it takes *user$_2$*'s computer to process the command for all *user$_2$*'s input commands in the subsequence except the last.

$V_1 = C_1+\sum D_{1,j}$: $C_1$ and $\sum D_{1,j}$ account for all components of $\sum R_i$ in the first subsequence. Based on equations 11.1, 11.2, and 14, we have:

$V_1 = 2d+(c_{1,1}+c_{2,1}-1)(w/p_2-w/p_1)+(c_{2,1}-c_{1,1}+1)t$ if $t-(w/p_2-w/p_1) \leq 0$ [Eq. 15.1]
$V_1 = 2d+c_{2,1}t+c_{2,1}(w/p_2-w/p_1)$ if $t-(w/p_2-w/p_1) > 0$ [Eq. 15.2]

$\sum R_i$ in Non-First Subsequences

$V_i = C_i + \sum D_{i,j}$ accounts for all components of $\sum R_i$ in subsequence *i*, where $i > 1$. The other subsequences are different from the first subsequence for the following reason. In the first subsequence, $user_2$'s computer is ready to process the first input command of $user_1$ in the subsequence as soon as it arrives. However, in other subsequences, $user_2$'s program replica may still be processing the last input command of the previous subsequence by the time the first input command of the current subsequence reaches it. As in Figure 2, this occurs if $w/p_2 - w/p_1 > 2d + t$, that is, if the difference in command processing times is greater than the time it takes for the last input of the previous subsequence to reach $user_1$'s computer, for $user_1$'s computer to process it, the think time before $user_1$ enters his first input command of the current subsequence, and the time it takes for this input to reach $user_2$'s computer. This additional delay increases the processing time for all non-first subsequences by $\delta_2 = w/p_2 - w/p_1 - 2d - t$.

Hence, the processing time equations 15.1 and 15.2 for the first subsequence can be generalized for non-first subsequences as follows:

$$V_i = 2d + (c_{1,i} + c_{2,i} - 1)(w/p_2 - w/p_1) + (c_{2,i} - c_{1,i} + 1)t \quad \text{if } t - (w/p_2 - w/p_1) \le 0 \text{ [Eq. 16.1]}$$

$$V_i = 2d + c_{2,i}t + c_{2,i}(w/p_2 - w/p_1) \quad \text{if } t - (w/p_2 - w/p_1) > 0 \text{ [Eq. 16.2]}$$

$$V_i = (c_{1,i} + c_{2,i})(w/p_2 - w/p_1) + (c_{2,i} - c_{1,i})t \quad \text{if } t - (w/p_2 - w/p_1) + 2d \le 0 \text{ [Eq. 16.3]}$$

We can now calculate $\sum L_i + \sum R_i$ which is the same as $\sum L_i + \sum V_i$.

$\sum L_i + \sum R_i$: Recall that $c_1$ and $c_2$ are the total number of commands by $user_1$ and $user_2$, respectively, *s* is the number of full subsequences, and *a* is the number of $user_1$'s answered commands. Then the task completion time for a replicated architecture, based on equations 6, 16.1, 16.2, and 16.3 is

$$task_{rep} = (c_1 + c_2)w/p_1 + 2sd + (a + c_2 - s)(w/p_2 - w/p_1) + (c_1 + c_2 - a + s - 1)t$$
$$\text{if } t - (w/p_2 - w/p_1) \le 0 \qquad \text{[Eq. 17.1]}$$

$$task_{rep} = (c_1 + c_2)w/p_1 + 2sd + c_2(w/p_2 - w/p_1) + (c_1 + c_2 - 1)t$$
$$\text{if } t - (w/p_2 - w/p_1) > 0 \qquad \text{[Eq. 17.2]}$$

$$task_{rep} = (c_1 + c_2)w/p_1 + 2d + (a + c_2 - 1)(w/p_2 - w/p_1) + (c_1 + c_2 - a)t$$
$$\text{if } t - (w/p_2 - w/p_1) + 2d \le 0 \qquad \text{[Eq. 17.3]}$$

In equation 17.3, $\delta_2$ is subtracted as it does not occur in the first subsequence.

## Feedthrough in Centralized Architectures

Recall that the feedthrough time for a command is defined as the time that elapses from the moment the command is input to the moment the non-inputting user sees its output. In centralized architectures, feedthrough depends on whether the inputting user is local or remote to the computer hosting the centralized program. If the *local* user provides the input, the *remote* user will see the output once the input command is processed and the output traverses the network. If the *remote* user provides the input, the *local* user will see the output once the input command reaches the local computer and the local computer processes it. Thus:

$\text{Feed}_{\text{CentTo}}U_1 = w/p_1+d$ if *user₁* hosts the program [Eq. 18.1]

$\text{Feed}_{\text{CentTo}}U_2 = w/p_2+d$ if *user₂* hosts the program [Eq. 18.2]

## Feedthrough in Replicated Architectures

Consider first feedthrough to commands input by the slower user, $user_2$. Such a command must traverse the network and be processed by the faster user's computer before the latter sees its output. Thus,

$\text{Feed}_{\text{RepFor}}U_2 = w/p_1+d$ [Eq 19.1]

The feedthrough time of commands input by the faster user, $user_1$, is more complicated because if $t < (w/p_2-w/p_1)$, the slower computer falls further behind the faster computer with each consecutive input entered by the faster user, as illustrated in Figure 2. In this case, consider, command, *j*, entered in the first subsequence. As this is the first subsequence, $user_2$'s computer processes the first command as soon as it arrives. Hence:

$\text{Feed}_{\text{RepFor}}U_1^{1,j} = d+w/p_2$ for $j = 1$ [Eq 19.2]

As $t < (w/p_2-w/p_1)$, the feedthrough will increase by $(w/p_2-w/p_1-t)$ for each subsequent command by $user_1$ in the subsequence. Hence:

$\text{Feed}_{\text{RepFor}}U_1^{1,j} = d+w/p_2+(j-1)(w/p_2-w/p_1-t)$ for $j \geq 1$ [Eq 19.3]

If $t \geq (w/p_2-w/p_1)$, the slow computer is ready to process each command by $user_1$ as soon as it arrives. Thus:

$\text{Feed}_{\text{RepFor}}U_1^{1,j} = d+w/p_2 \;\; j \geq 1$ [Eq 19.4]

Recall that there are two cases to consider for the non-first subsequences. If $w/p_2-w/p_1 \leq 2d+t$, the slow computer is ready to process the first command in the subsequence as soon as it arrives. In this case, the feedthrough equations given above for the first subsequence apply to all subsequences. Otherwise, the term $w/p_2-w/p_1-2d-t$ is added to the equations given above for the first subsequence.

## Formal Analysis Validation

We have given above both mathematical proofs and intuition for justifying the performance model. In addition, it is important to back these with experimental results that validate it for a large number of values of collaboration parameters. Ideally, these experiments should also show its practicality. Several approaches could be used to gather the experimental data.

- Live interaction: Under this approach, pairs of users would perform a collaborative task multiple times as the architecture and system parameters are varied in a controlled manner each time.

- Actual logs: Another approach is to use logs of actual collaborations and assume that these are independent of the system parameters such as architecture, machines used, and network delays. These logs can then be replayed under different values of system parameters.
- Synthetic logs: With this approach, the user logs can be created by varying the user parameters using some mathematical distribution such as Poisson's.

Since users cannot be relied upon to perform the same sequence of actions and have the same think times in different collaborative sessions, the live interaction approach is impractical. The other two approaches require a large number of logs to ensure that a wide range of values for user parameters are covered. This is not a problem for synthetic logs, but such logs do not address the practicality concern as it is not clear parameter values based on mathematical distributions represent reality. Logs of actual interaction are not provided in any public database and we were unsuccessful in obtaining them from researchers who we knew had logged their collaboration tasks. Thus to use the actual-log approach, we would have to gather a large number of actual logs ourselves, which is beyond the scope of our work: the analytical model is our primary contribution and the experiments are addressed mainly to validate the model. In other fields such as real-time systems where benchmarks are not widely available, it is customary to resort to the synthetic-log approach to validate new theoretical results. We did a little better by using a hybrid of the synthetic and actual log approaches. We recorded a small number (8) of actual logs to obtain realistic values of some user parameters and then used these values to create a large number (30) of synthetic logs that we then replayed in the actual experiments using different architectures and system parameters.

We used the same program for recording the actual logs and replaying the synthetic logs. The program is the distributed checkers program used by Chung and Dewan (2001, 2004) which allows a group of users to play against the computer. We chose this program for two reasons. First, it is a computer-intensive task, allowing us to validate the effect of processing time differences. Second, the user study participants knew the game rules, so no user training was needed.

Recall that we assume that an input command takes exactly $w$ CPU cycles to be processed. In Checkers, a user's move consists of two actions: picking up and putting down a piece. To make our response and feedthrough measurements valid, we group the multiple input commands for a single move into a single input command. Also, the computer calculation of the next move depends on the piece positions and is hence not constant. Thus, we report the average response and feedthrough times over all the moves in a single game.

We focus on actor-observer interaction mode in which one user, the actor, makes all the inputs, which are at the end acknowledged by the other user, the observer. The acknowledgement is needed to tell the actor that the observer has seen all the moves and they can proceed to their next task (e.g. post mortem of the game.) Focusing on the actor-observer allows us to show practical results in the limited space available while addressing many common collaborative tasks, such as an expert demonstrating to others or a pupil being tested by a tutor.

To approximate the think times and logs for the actor-observer mode, we gathered actual user logs in which a single user played against the computer. Table I shows the values of user parameters obtained from these studies, which were used in the synthetic logs.

| Number of moves | | | Think Time (s) | | |
|---|---|---|---|---|---|
| Min | Med | Max | Min | Med | Max |
| 21 | 44 | 72 | 0.11 | 4.5 | 41.2 |

Table I. Measured User Parameter Values Used In Synthetic Logs

The system parameters, processing powers and network delays, also have to be realistic. We used two computers, a Pentium 1.5M laptop and a P2 400Mhz desktop, which have a processing power difference that can be expected when two users collaborate. Both computers are connected on a local LAN. Based on Chung's and Dewan's experiments, we added 72, 162 and 370 ms to the LAN delays to estimate half the round-trip time from a U.S. East Coast LAN-connected computer to a German LAN-connected computer, German modem-connected computer, and Indian LAN-connected computer, respectively. As LAN delays vary during an experiment, we performed it ten times and report the average performances for these ten trials. Our measured numbers are consistent with our model. We do not have space to give all of our measurements. We report a sample of them next when we discuss and validate the new predictions made by our model.

## Applications of our Model

The application of our work consists of (1) explaining previous experimental results, and (2) making new predictions.

### Explaining Architecture Performance Results

Chung and Dewan showed that (a) low network latencies favor a centralized architecture and (b) asymmetric processing powers favor a centralized architecture. These results were interesting because they went against the common assumption that a replicated architecture always outperforms a centralized one. In their experiments they used a think time of zero and a centralized case in which the faster computer executed the program. Under these conditions, equation 5.1 applies for the centralized architecture task completion time, and equations 17.1 and 17.3 apply to the replicated architecture. For brevity, we consider only 17.3

here. To compare the task completion times of the two architectures, we can subtract equation 17.3 from equation 5.1. A negative result means that the centralized architecture has a better task completion time, and vice versa.

$task_{cent}U_1\text{-}task_{rep} = 2d(c_2\text{-}1)\text{-}(a+c_2\text{-}1)(w/p_2\text{-}w/p_1)$

Assume that $c_2$, the number of input commands entered by *$user_2$*, is always greater than 1. Also assume that the number of *$user_1$*'s answered input commands, *a*, is greater than or equal to 0. Thus, the term $2d(c_2\text{-}1)$ will increase as *d*, the network delay, increases, which favors the replicated architecture. On the other hand, the term $\text{-}(a+c_2\text{-}1)(w/p_2\text{-}w/p_1)$ in the same equation will become more negative as the processing power difference, $w/p_2\text{-}w/p_1$, increases, which favors centralizing the program to *$user_1$*'s machine. This is consistent with the results by Chung and Dewan. Next we consider new predictions made by our analytical model. In all of the cases we consider below, we assume the actor-observer mode defined earlier.

Choosing the Placement of the Centralized Program

Sometimes replication is not an option – a user may be bound to a centralized system or the shared program cannot be executed correctly in the replicated architecture. Our model, somewhat counter-intuitively, predicts that centralizing the program on the slower computer may give better task completion and response times but worse feedthrough times than centralizing to the faster computer.

This occurs when the actor is the user on the slow computer. The relevant task completion time equations are 5.1 and 5.2 and the relevant response time equations are 1.3 and 1.4. Consider the task completion and response time difference equations for the two centralized architectures:

$task_{cent}U_1\text{-}task_{cent}U_2$
$= 2d(c_2\text{-}c_1)\text{-}(c_1+c_2)(w/p_2\text{-}w/p_1)$

$Resp_{Cent}U_2[1.4]\text{-}Resp_{Cent}U_2[1.3]$
$= 2d\text{-}(w/p_2\text{-}w/p_1)$

We can set $c_1 = 1$ because the observer, *$user_1$* in this case, provides one input at the end to acknowledge end of collaboration. We assume that $c_2$, the number of input commands entered by *$user_2$*, is always greater than 1. Thus in the task completion time difference equation, the term, $2(c_2\text{-}1)d$, will increase as *d*, the network delay, increases, which favors centralizing the program to *$user_2$*'s slower machine. However, since $c_2 > 1$, the $(1+c_2)(w/p_2\text{-}w/p_1)$ term in the equation will increase with the processing time difference, $w/p_2\text{-}w/p_1$, which favors centralizing the program to *$user_1$*'s machine. The same conditions apply to the response time difference. For feedthrough, however, centralizing on the fast computer gives a lower value, as shown below. In both centralized architectures, either *$user_1$*'s input command or its output will traverse the network before *$user_2$* sees the output. Therefore, the feedthrough will be lower when centralizing to *$user_1$*'s faster computer as it processes input commands faster than *$user_2$*'s slower computer.

$Feed_{cent}U_1\text{-}\ Feed_{cent}U_2 = w/p_1+d\text{-}w/p_2\text{-}d = \text{-}(w/p_2\text{-}w/p_1)$

To experimentally validate this scenario, we used the median observed think time of 4.5s. The results in Figure 4 confirm our analyses. As we see, centralizing on the slow computer offers relative gains of as much as 69% for response time and 10% for the task completion time. As expected, when think times dominate the task completion time, the relative task completion time difference is not large.

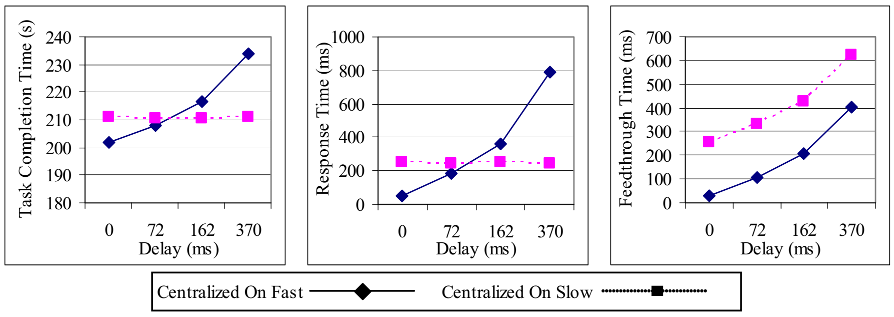

Figure 4. Task Completion, Response, and Feedthrough Times of the Slow Actor

## Centralized and Replicated Task Completion Times

Our model also predicts that in certain collaboration modes, the task completion time advantage one architecture has over another can be significant. In particular, a centralized architecture with the faster user's computer hosting may enjoy such an advantage over a replicated architecture when think times are low and the user with the faster computer, *user$_1$*, is the actor. The relevant equations are 5.1, 17.1, 17.2, and 17.3. We consider 5.1 and 17.1 only which give the task completion time difference as

$$\text{task}_{\text{cent}}U_1\text{-task}_{\text{rep}} = 2d(c_2\text{-}s)\text{-}(a+c_2\text{-}s)(w/p_2\text{-}w/p_1)+(a\text{-}s)t$$

As before, we can set $c_2 = 1$ because the observer, *user$_2$* in this case, provides one input at the end of the collaboration. We assume that $c_1$, the number of input commands entered by the actor, *user$_1$*, is always greater than 1, all of which are answered. Hence, $c_1 = a > 1$. Because the collaboration consists of one full subsequence, $s = 1$. Thus, $2(c_2\text{-}s)d = 0$ and $(a+c_2\text{-}s) = a$. We assume that $t$, the think time, is less than $(w/p_2\text{-}w/p_1)$, the processing time difference. Thus, the task completion time difference, which equals $\text{-}a(w/p_2\text{-}w/p_1)+(a\text{-}1)t$, is negative. Since the faster user inputs all the commands but one, the network delays are not a factor. Hence, processing the input commands only on the faster computer is better than replicating because the slower computer falls further behind the faster one with each input command which increases the task completion time as predicted by the model.

To experimentally validate this scenario, we used the minimum observed think time of 110ms as it needed to be less than the time difference for processing an input command to make the above equations hold. We show the experiment results with LAN delays (0ms). The results for other delays are consistent but are omitted for brevity reasons. Figure 5 confirms our analysis. It shows that the centralized architecture can be as much as 6.3s faster or as much as 58% quicker in completing the task than the replicated architecture.

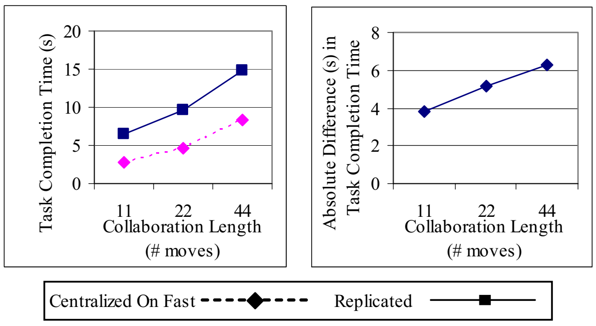

Figure 5. Task Completion, Response, and Feedthrough Times of the Slow Actor

## Collaboration Length Effect on Task Completion and Feedthrough Times

Our model also predicts that for certain collaboration conditions, the advantage in task completion and feedthrough times one architecture has over another gets magnified as the length of the collaboration increases.

As above, this occurs when think times are low, that is, $t < w/p_2\text{-}w/p_1$, and the user with the faster computer, $user_1$, is the actor. In the above analysis, we show that in this collaboration mode, the centralized architecture with $user_1$'s computer hosting completes the task in $a(w/p_2\text{-}w/p_1)\text{-}(a\text{-}1)t$ time faster than the replicated architecture. Since $a$, the number of answered $user_1$'s input commands, increases with collaboration length, this difference gets magnified according to the above task completion time difference equation. Intuitively, the time that elapses from the moment the faster computer processes an input command and to the moment the slower computer processes the same command increases between consecutive inputs by the actor because the slower computer falls further behind the faster one with each input command. Figure 5 verifies our analysis.

Consider now the feedthrough times. Since the entire collaboration consists of consecutive actor's input commands followed by a single input command from the observer, there is only one subsequence in the collaboration. Thus, the relevant feedthrough time equations are equations 18.1 and 19.3. According to these equations, the feedthrough time difference is:

$$\mathrm{Feed}_{\mathrm{CentTo}}U_1\text{-}\mathrm{Feed}_{\mathrm{RepFor}}U_1{}^{1,j} = \text{-}(w/p_2\text{-}w/p_1)\text{-}(j\text{-}1)(w/p_2\text{-}w/p_1\text{-}t) \text{ for } j \geq 1$$

The maximum value of $j$ is $a$, the number of $user_1$'s answered input commands. Consider the feedthrough of $user_1$'s last input command. With the same reasoning as above, we can argue that the feedthrough of $user_1$'s last input command will, in the replicated case, increase with collaboration length as predicted by the model.

We validate this indirectly using task completion time results in Figure 5. The time *user*$_2$ views the output to *user*$_1$'s last input command is exactly the task completion time minus the time it takes to process *user*$_2$'s only input. Since the previous result showed that the absolute task completion time difference increases with collaboration length in favor of the centralized architecture, we can conclude that the feedthrough time of *user*$_1$'s last input message will behave the same.

### Other Predictions

Our model makes the following additional predictions. In some cases, the replicated architecture optimizes the feedback time but not the task completion time. In other cases, centralizing to the slower computer may offer equivalent task completion and feedback times as replicating. Moreover, as the think time increases, the architecture choice makes no significant difference to the task completion time (though it does influence the response and feedthrough times). This happens for two reasons. The obvious one is that think times dominate the task completion time. The more interesting one we found is that the users who had high think times also had smaller number of moves. As we saw above, relative difference in task completion times of replicated and centralized architectures decreases as input sequence length decreases. We do not deduce or validate these predictions through experiments because of lack of space.

## Conclusions and Future Work

This paper makes several contributions.

First, the analysis offers a better understanding of the event flow in centralized and replicated architectures and explains previous experimental results.

Second, the analysis provides several new conclusions regarding the performances of two-user architectures such as (a) centralizing the application on the slower machine is sometimes the optimal solution, (b) centralizing the application on the faster machine can be better than replicating, and (c) as the duration of the collaboration increases, the difference in performances of centralized and replicated architectures gets magnified.

Third, the analysis provides guidance for users with varying degrees of choice regarding the collaborative systems they use. For users who are bound to a particular collaboration system, we offer an analysis of how changes to the collaboration parameters can help improve performance. Given a choice of systems that support a single collaborative architecture or that bind a session to an architecture at session start time, we provide a way to make a better decision about which system or architecture to select to obtain optimal performance. When a system supports runtime architecture changes, we help decide which architecture to use as the user and system parameters change.

Lastly, as secondary contributions we report results of our user studies. The logs we collected from actual usage give values of user parameters such as think times and number of user actions that are relevant to performance. The performances we report from the replays of synthetic logs generated from the user parameters constitute new empirical results in this area.

Further work is needed to make and verify additional predictions based on our analyses, use actual user logs for making measurements, relax assumptions made in our analysis, and build a system module that automatically changes the architecture based on the analysis and current values user and system parameters. We hope this first-cut at a formal model will be a catalyst for such work.

## Acknowledgements

This research was funded in part by *Microsoft* and NSF grants ANI 0229998, EIA 03-03590, and IIS 0312328. We also thank Tynia Yang, Ankur Agiwal, John Calandrino, Enes Junuzovic, Jason Sewall, Stephen Titus, and Ben Wilde, our user study participants, who volunteered their time on a very short notice.

## References

Ahuja, S., Ensor, J.R., Lucco, S.E. (1990): 'A Comparison of Application Sharing Mechanisms in Real-time Desktop Conferencing Systems', *Proceedings of the conference on Office Information Systems*, 1990, pp: 238-248.

Chung, G. and Dewan P. (2001): 'Flexible Support for Application-Sharing Architecture', *Proceedings of the European Conference on Computer Supported Cooperative Work*, 2001.

Chung, G and Dewan P. (2004): 'Towards Dynamic Collaboration Architectures', *Proceedings of the 2004 ACM conference on Computer Supported Cooperative Work*, 2004.

Danskin, J., Hanrahan, P. (1994): 'Profiling the X Protocol', *Proceedings of the 1994 ACM Conference on Measurement and Modeling of Computer Systems*, 1994, pp: 272-273.

Droms, R., Dyksen, W. (1990): 'Performance Measures of the X Window System Communication Protocol', *Software – Practice and Experience (SPE)*, vol. 20, no. S2, 1990, pp: 119-136.

Dyck, J., Gutwin, C. Subramanian, S., Fedak, C. (2004): 'High-performance Telepointers', *Proceedings of the 2004 ACM conference on Computer Supported Cooperative Work*, 2004.

NEC 2000: 'Windows 2000 Terminal Services Capacity and Scaling', *NEC*, 2000 .

Nieh, J., Yang, S. and Novik, N. (2000): 'A Comparison of Thin Client Computing Architectures', *Technical Report CUCS-022-00 Columbia University*, November 2000.

Richardson, T., Stafford-Fraser, Q., Wood, K., Hopper, A. (1998): 'Virtual Network Computing', *IEEE Internet Computing*, vol. 2, no. 1, January/February 1998, pp. 33-38.

Wong, A., Seltzer, M. (2000): 'Evaluating Windows NT Terminal Server Performance', *Proceedings of the 3rd USENIX Windows NT Symposium*, July 1999, pg: 145-154.