Measuring and Understanding User Comfort With Resource Borrowing

Ashish Gupta Bin Lin Peter A. Dinda
{ashish,binlin,pdinda}@cs.northwestern.edu
Department of Computer Science, Northwestern University

Abstract

Resource borrowing is a common underlying approach in grid computing and thin-client computing. In both cases, external processes borrow resources that would otherwise be delivered to the interactive processes of end-users, creating contention that slows these processes and decreases the comfort of the end-users. How resource borrowing and user comfort are related is not well understood and thus resource borrowing tends to be extremely conservative. To address this lack of understanding, we have developed a sophisticated distributed application for directly measuring user comfort with the borrowing of CPU time, memory space, and disk bandwidth. Using this tool, we have conducted a controlled user study with qualitative and quantitative results that are of direct interest to the designers of grid and thin-client systems. We have found that resource borrowing can be quite aggressive without creating user discomfort, particularly in the case of memory and disk. We also describe an on-going Internet-wide study using our tool.

1 Introduction

Many widely used distributed computing platforms and applications use available resources on existing host computers, exploiting the fact that most of these systems are dramatically under-utilized [20, 5, 1], a technique we refer to as *resource borrowing.* Examples in scientific computing include Condor [19, 11], Entropia [3], SETI@Home [30], Protein Folding at Home [18], DESChall [4], and the Google Toolbar [12]. The majority of such systems and applications run on Microsoft Windows platforms, and they are deployed on hundreds of thousands (SETI@Home) to millions (Google) of hosts. Examples in peer-to-peer content distribution systems include commercial tools such as Kazaa [28] and Gnutella [22], as well as academic projects [29, 23, 15, 14]. Some of these systems are deployed on millions of Windows hosts.

Effort sponsored by the National Science Foundation under Grants ANI-0093221, ACI-0112891, ANI-0301108, EIA-0130869, and EIA-0224449. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation (NSF).

There have also been numerous proposals to consolidate desktop computers and servers onto clusters [26, 10] to simplify their administration and exploit their dramatic underutilization to make computing more economical. Interactive users would then use thin clients [27] to access their processes. In effect, each user will see a slowed machine due to resources borrowed for other users.

In both cases, several fundamental questions arise about user's interaction with resource borrowing:

1 What level of resource borrowing leads to user discomfort for a significant fraction of users?
2 How does the level depend on which resource or combination of resources is borrowed?
3 How does the level depend on the user's context (the foreground task)?
4 How does the level depend on the user, factoring out context?
5 How does the level depend on the time dynamics of resource borrowing?
6 How does the level depend on the raw power of the host?

Current systems assume very conservative answers to these questions because if they cause the user to feel that the machine is slower than is desirable, the user is likely to disable them. For example, the default behavior in Condor, Sprite [7] and SETI@Home is to execute only when they are quite sure the user is away, when the screen saver has been activated. Other systems run at a very low priority, or they simply ask the user to specify constraints on resource borrowing, something that few ordinary users understand. If less conservative resource borrowing does not lead to significantly increased user discomfort, the performance of current systems could be increased through techniques such as linger-longer scheduling [24].

There is indirect evidence that resource borrowing need not be especially conservative in that many users are willing to install peer-to-peer tools, and run services such as Microsoft's IIS and Kazaa, Gnutella, etc, on their desktop computers. Certainly, the extremely low utilization of available CPU cycles, which has held true for over 10 years, suggests that if the "right" cycles were used, those "in between" the cycles the user is using, there would be little for the user to perceive. The priority-based schedulers of modern operating systems approximate this.

Despite indirect evidence, there exist no quantitative, empirical measurements that could be used to answer the above questions. This may seem surprising to some readers, as this would appear to be an excellent problem in human-computer interaction or psychology. However, the work in those areas has concentrated on the impact of latency on user-perceived utility of the system [17, 8], and on user frustration to different user interfaces [16, 21]. Within the systems community, related work has examined the performance of end-user operating systems using latency as opposed to throughput [9], and suggested models for interactive user workload [2].

In response to this lack of information, we have developed a system, the Understanding User Comfort System (UUCS). UUCS is a distributed Windows application similar to SETI@Home in design. A UUCS client emulates resource borrowing of CPU, memory and disk on the user's machine in a controlled manner encoded in a *testcase* provided by a server. The user operates the machine as normal, but if his interactivity gets affected, he may express discomfort by clicking on an icon or pressing a hot-key. Resource borrowing stops immediately if discomfort is expressed or when the testcase is finished. The point of discomfort, if any, is returned to the server along with contextual information. By analyzing the results of applying a particular set of testcases to a particular set of users, we can qualitatively and quantitatively characterize user comfort. Note that a background application has full control over the resources that it borrows, but a mapping between resource borrowing and interactivity metrics like system latency or jitter is difficult to obtain. This study is intended to directly study the end-to-end relationship between resource borrowing and user comfort.

Using UUCS, which we describe in detail in Section 2, we have conducted a carefully controlled user study at Northwestern University to address the questions raised earlier. We describe this study and its results in detail in Section 3, while Section 4 describes our ongoing Internet-scale study. Finally, in Section 5, we provide advice to implementors based on our study results. More information about the UUCS system can be found in a separate technical report [13]. To examine our software, flyers and questionnaires, or to participate visit http://comfort.cs.northwestern.edu.

2 System design

UUCS consists of a server and a client, as shown in Figure 1. Both are Windows applications that store testcases and results on permanent storage in text files. A testcase contains functions that describe how to "exercise" a collection of resources.

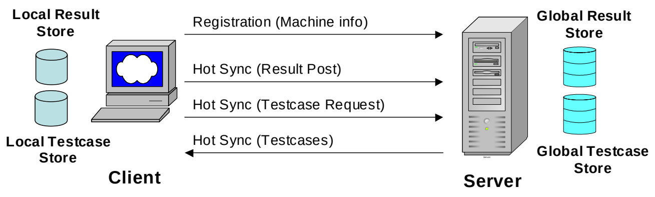


Figure 1. Structure of UUCS.

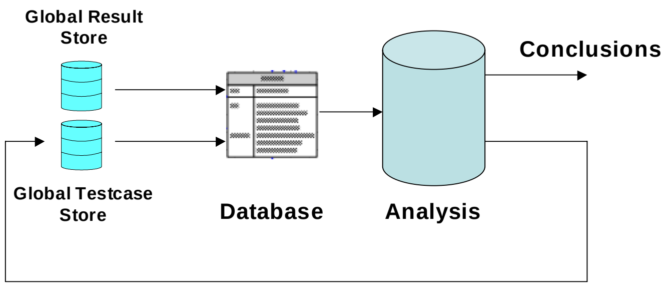


Figure 2. The Analysis Phase.

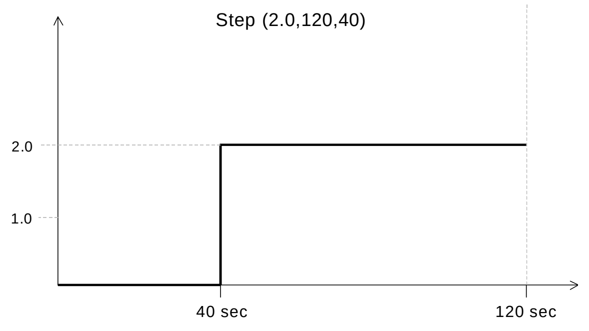
Using its local testcase and result stores, the client can operate disconnected from the server. There are two interactions between the two, both initiated by the client. When the client is initially run, it registers with the server, providing it with a detailed snapshot of the hardware and software of the client machine, and allowing the server to associate a globally unique identifier with the client. Subsequently, at particularly selected times when the client is connected to the network, the client initiates a "hot sync" with the server. New testcases, which can be added to the server at any time, are downloaded by the client, while new results are uploaded back to the server.

Hot syncing operates at user-defined intervals and acquires a growing random sample of testcases from the server. This, combined with local random choice of testcases and Poisson arrivals of testcase execution, is designed to make a collection of clients execute a random sample with respect to testcases, users, and times. This is the mode of operation that we use in our Internet-wide study. A UUCS client can also be configured to behave deterministically, executing a predefined set of commands from a local file. We use this feature in our controlled study.

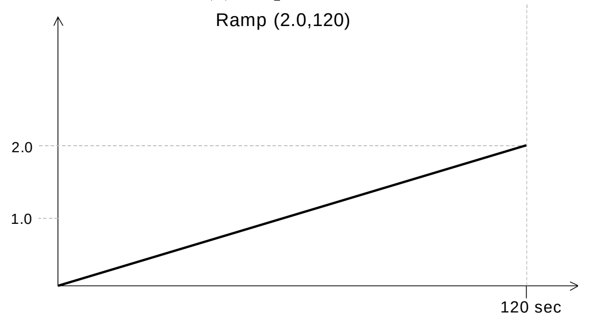
In addition to the client and server, we have developed a set of tools for creating, viewing, and manipulating testcases, and for importing testcase results into a database (Figure 2). An additional set of tools is then used to analyze the results and guide us to other interesting testcases.

Name	Description
$step(x,t,b)$	contention of zero to time b, then x to time t
$ramp(x,t)$	ramp from zero to x over times 0 to t
sin	sine wave
saw	sawtooth wave
$expexp$	Poisson arrivals of exponential-sized jobs (M/M/1)
$exppar$	Poisson arrivals of Pareto-sized jobs (M/G/1)

Figure 3. Exercise functions.



(a) Step testcase



(b) Ramp testcase

Figure 4. Step and ramp exercise functions.

2.1 Testcases and exercise functions

Testcases encode the details of resource borrowing for various resources. A testcase consists of a unique identifier, a sample rate, and a collection of *exercise functions*, one for each resource that will be used during the execution of the testcase (the *run*). An exercise function is a vector of values representing a time series sampled at the specified rate. Each value indicates the level of contention (the extent of resource borrowing, described in Section 2.2) for a resource at the corresponding time into the testcase. For example, consider a sample rate of 1 Hz, and the vector $[0, 0.5, 1.0, 1.5, 2.0]$ for the CPU resource. This exercise function persists from 0 to 5 seconds from the start of the testcase. From 3 to 4 seconds into the testcase, it indicates that contention of 1.5 should be created and subsequently 2.0 in the next second.

Our testcase tools let us generate testcases containing exercise functions of many different kinds, as summarized in Figure 3. In our controlled study, we use a small set of testcases that contain $step$ or $ramp$ exercise functions with different parameters for single resources. Figure 4 shows examples for $step(2.0, 120, 40)$ and $ramp(2.0, 120)$ respectively. In our Internet-wide study, we currently have over 2000 testcases. The large number of testcases are designed to study a wide variety of resource borrowing behavior with a range of parameters for each exercise function type, predominantly from the M/M/1 and M/G/1 models. Note that each testcase may be run multiple times by different users.

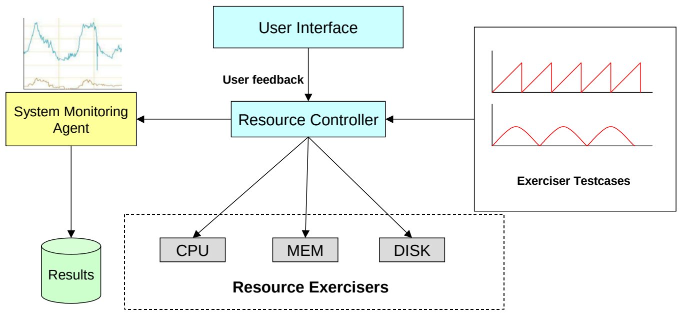


Figure 5. Client design.

2.2 Resource exercisers

Resource exercisers are important components of the client that apply the contention described by an exercise function. There are three exercisers: CPU, memory, and disk. They run at the same priority as other threads.

The CPU exerciser implements time-based playback of the exercise function, as we describe and evaluate in detail in earlier work [6]. Consider the previous example, where we are asked to create a contention of 1.5 from 3 to 4 seconds. Two threads with carefully calibrated busy-wait loops will execute for one second. These loops split the one second interval into a number of subintervals, whose duration is computed by calibration, each larger than the scheduling resolution of the machine. The first loop will only execute busy subintervals,with no sleeps. The second executes busy subintervals with probability 0.5, calling ::Sleep in other subintervals. The stochastic borrowing is intended to emulate a fluid model, but is limited by the time quantum of the underlying scheduling mechanism. This depends on the OS and its version. The effect is that if there is another busy thread in the system (the display loop of a first person shooter game, for example), that thread will execute at a rate $1/(1.5 + 1) = 40$ % that of the maximum possible rate on the system, the CPU exerciser having borrowed 60% of the CPU. This exerciser is experimentally verified to a contention level of 10 for equal priority threads.

The disk exerciser operates nearly identically to the CPU exerciser, except that its goal is to create contention for disk bandwidth. The busy operation here is a random seek in a large file (2x the memory of the machine) followed by a write of a random amount of data. The write is forced to be write-through with respect to the windows buffer cache and synced with respect to the disk controller. Contention here has the effect of slowing down the I/O of another I/O-busy thread similarly to the CPU exerciser. This exerciser is experimentally verified to a contention level of 7 for equal priority threads.

The memory exerciser is considerably different. It interprets contention as the fraction of physical memory it should attempt to allocate. It keeps a pool of allocated pages equal to the size of physical memory in the machine and then touches the fraction corresponding to the contention level with a high frequency, making its working set size inflate to that fraction of the physical memory. This ensures borrowing of physical memory by the desired amount. We avoid contention levels greater than one because this immediately results in thrashing which is not only very irritating to all users (as it affects interactivity drastically), but also very difficult to stop punctually.

Using the network can also lead to user discomfort. We developed several variants of a network exerciser (described in our technical report), but all create a significant impact beyond the client machine. For this reason, we did not study the effect of network resource borrowing. We plan to do so in the future.

2.3 Testcase execution and system monitoring

When a testcase is executed, the appropriate exercisers are started, passed their exercise functions, synchronized, and then let run. A high priority GUI thread watches for clicks or hot-key strokes. If this occurs, the exercisers are immediately stopped and their resources released. The testcase run is over when user expresses discomfort feedback or the exercise functions are exhausted without any feedback. A considerable amount of information is stored as the result of the testcase run, including CPU, memory and Disk load measurements for entire duration of the testcase, system processes information, foreground process information, etc. A number of technical articles, discussed in our technical report, were very useful in the development of the resource exercisers and monitors. Figure 5 illustrates the structure of the UUCS client.

The detailed registration information for the client along with collected data is stored in text-based form for later communication back to the server. For the remainder of this paper, we use the following data:

- Whether the testcase run was terminated due to user feedback or testcase exhaustion,
- The time offset into the testcase at which irritation or exhaustion was reported, and

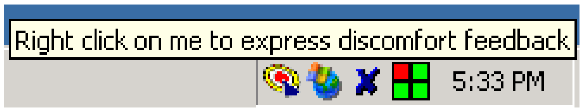

Right click on me to express discomfort feedback

Figure 6. Basic client interface. A menu and full interface are also available.

Machine Configuration	
Hardware Configuration	2.0 GHz P4, 512 MB, 80 GB. Dell Optiplex GX270, 17 in monitor 100 Mbps Ethernet
Operating System	Windows XP
Applications Installed	Word 2002, Powerpoint 2002, IE 6, Quake III

Figure 7. Machine configuration.

- The last five contention values used in each exercise function at the point of user feedback.

2.4 Client interface

Figure 6 shows the most basic graphical interface of the UUCS client, as used in our study. Here, the user can only request discomfort, either by clicking on the tray icon or pressing a hot-key (F11).

3 Controlled study

Using the UUCS, we ran a controlled study at Northwestern to help us answer the questions posed in the introduction. Compared to our ongoing Internet-wide study, this study had a limited number of participants, but because of the careful control of factors, we can directly address many of the questions.

3.1 Perspective of the user

The 33 users in our study consisted primarily of graduate students and undergraduates from the Northwestern engineering departments. We advertised for participants via flyers and email. Each user was given $15 for participating. The machine configuration used in the controlled study is shown in Figure 7. Two such machines were set up in separate, private environments. The duration of the study for each user was 84 minutes. The user:

- Filled out a questionnaire. The key questions were user self-evaluations as “Power User”, “Typical User”, or “Beginner” for use of PCs, Windows, Word, Powerpoint, Internet Explorer, and Quake. (5 minutes).
- Read a one page handout. (5 minutes).

No.	Resource	Type	Word Parameters	Powerpoint	Internet Explorer	Quake
1	CPU	Ramp	7.0,120	2.0,120	2.0,120	1.3,120
2	Blank					
3	Disk	Ramp	7.0,120	8.0,120	5.0,120	5.0,120
4	Memory	Ramp	1.0,120	1.0,120	1.0,120	1.0,120
5	CPU	Step	5.5,120,40	0.98,120,40	1.0,120,40	0.5,120,40
6	Disk	Step	5.0,120,40	6.0,120,40	4.0,120,40	5.0,120,40
7	Blank					
8	Memory	Step	1.0,120,40	1.0,120,40	1.0,120,40	1.0,120,40

Figure 8. Testcase descriptions for the 4 tasks (given in random order).

- Acclimatized themselves to the performance of our machine by using the above applications. (10 minutes).
- Performed the following tasks:
    - Word processing using Microsoft Word (16 minutes): Each user typed in a non-technical document with limited formatting. [1]
    - Presentation making using Microsoft Powerpoint (16 minutes): Each user duplicated a presentation consisting of complex diagrams involving drawing and labeling from a hard copy of a sample presentation.
    - Browsing and research with Internet Explorer (16 minutes): Each user was assigned a news web site and asked to read the first paragraphs of the main news stories. Based on this, they searched for related material and saved it. This task involved multiple application windows.
    - Playing Quake III (16 minutes): Quake III is a well known first-person shooter game. There were no constraints on user's gameplay. This is our most resource intensive application.

These applications are intended to represent typical user tasks. As the user performed the tasks, the UUCS client executed in the background and ran specific testcases. It recorded all the system and contextual information as well as the user feedbacks, which were later used to generate the results. The user is aware of a background resource borrowing process but unaware of any details, and the process remains invisible to the user.

Note that our control study involves a self-selected sample, which, although we have mitigated the effects, mean that we must be careful when generalizing from it. However, anecdotal evidence suggests that this group is more sensitive to resource borrowing than others, and thus our results are likely to be conservative.

3.2 Testcase details

Our testcases were designed to help us address the questions in the introduction. They were either of the type ramp or step (Section 2.1), or blank. Blank testcases allow us to test for the background level of discomfort, while ramps allow us to test user tolerance to borrowing. Steps combined with ramps are used to test for sensitivity to one element of time dynamics. Each task had 8 associated testcases, each 2 minutes long. They are run in a random order for each 16-minute task. We call the execution of a testcase during a specific task by a specific user a *run*.

The regions of resource usage where interactivity is affected are different for each task. For example, in Word very high values of CPU contention (around 3) are needed to affect interactivity at all, while in Quake, CPU contention values in the region of 0.2 to 1.2 cause drastic effects. To observe the onset of discomfort, the parameters for the testcases for each task had to be chosen carefully. This calibration was done by having one of the authors use the applications while running a large number of testcases with different parameters, selecting those testcases which affected interactivity. Figure 8 shows the specific testcases used for each task.

It is important to point out that our calibration procedure was subjective. It could have been the case that our testcases were too aggressive or too lax, affecting too many or too few users. However, our results suggest that we have captured a wide range of behavior. Figure 9 counts the runs, grouped by the task, whether the testcase was blank or not, and whether the user expressed discomfort or did not react (testcase exhausted). As we can see, there were few reactions to blank testcases, and most, but not all, non-blank testcases caused discomfort at some level.

3.3 Results

We now address the questions posed in the introduction using empirical cumulative distribution functions and informal factor analysis. We describe the results below, along with additional observations.

[1] Even in a word processor, the space of interactions can be very large. Here we mainly cover typing and saving the document. Drawing figures is covered in the Powerpoint task.

Total		
	Non-Blank testcases	Blank
Discomforted	295	33
Exhausted	47	212
MS Word		
Discomforted	48	0
Exhausted	20	59
Prob of discomfort from blank testcase		0.00
MS Powerpoint		
Discomforted	71	0
Exhausted	4	60
Prob of discomfort from blank testcase		0.00
Internet Explorer		
Discomforted	50	14
Exhausted	17	50
Prob of discomfort from blank testcase		0.22
Quake		
Discomforted	126	19
Exhausted	6	43
Prob of discomfort from blank testcase		0.30

Figure 9. Breakdown of runs.

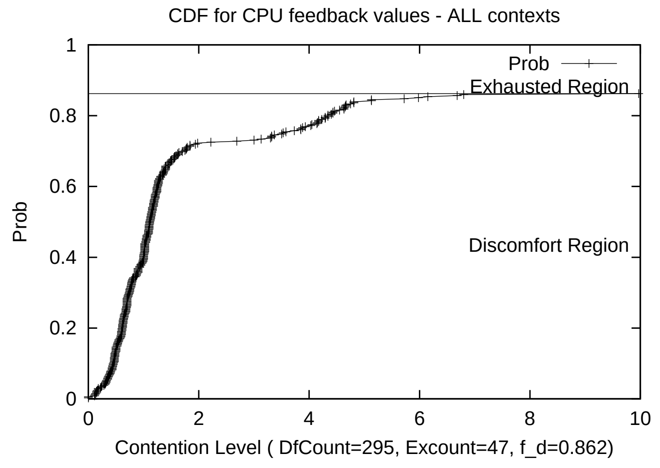


Figure 10. CDF of discomfort for CPU.

3.3.1 What level of resource borrowing leads to user discomfort for a significant fraction of users?

From the perspective of an implementor this is a key question. We can answer this question using cumulative probability distributions (CDF) derived from running our ramp testcases, aggregated across contexts to convey a general view of each resource.

Figures 10-12 show CDFs for CPU, Memory and Disk aggregated over all the tasks. The horizontal axis is the level of contention for each resource. The vertical axis is the cumulative fraction of users discomforted. As the level of borrowing increases, users' interactivity is increasingly likely to be affected. This is the *discomfort region*. Some users do not become discomforted in the range of levels explored. We refer to this as the *exhausted region.* The graph is labeled with the number of runs that ended in discomfort ($DfCount$) and exhaustion ($ExCount$). There is also some probability that a user will feel discomforted even when no resource borrowing (blank testcase) is occurring. We refer to this as the *noise floor* and it is reflected in Figure 9. To make our discussion easier, we derive three metrics from the CDFs. The first is f_d, the fraction of testcases which provoke discomfort, $f_d = \frac{DfCount}{DfCount+ExCount}$. A low value of f_d indicates that the range of contention applied in that context for resource borrowing doesn't affect interactivity significantly.

The second metric is $c_{0.05}$, the contention level that discomforts 5% of the users. This is the 5th-percentile of the CDFs. This value is of particular interest to implementors as it provides them with a level that discomforts only a tiny fraction of users. Any other percentile can also be found from these CDFs.

The third metric is c_a, the average contention level at which discomfort occurs. This is useful in comparing classes of users. Figures 14, 15, and 16 present the metrics.

Figure 10 shows the CDF for CPU borrowing. Notice that even at CPU contention levels of 10, more than 10% of users do not become irritated. More importantly, we can reach contention levels of 0.35 while irritating fewer than 5% of the users ($c_{0.05,cpu} \simeq 0.35$). This corresponds to consuming 35% of the processor when there are no competing threads.

Figure 11 shows the CDF for memory. Notice that almost 80% of users are unfazed even when nearly all their memory is consumed ($f_d = 0.21$). Furthermore, aggre-

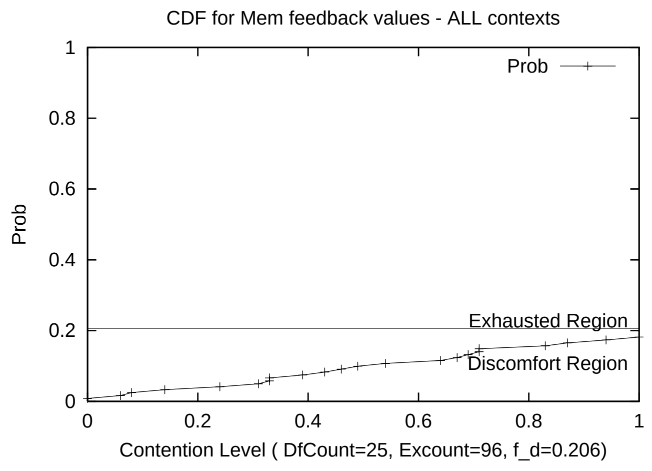


Figure 11. CDF of discomfort for Memory.

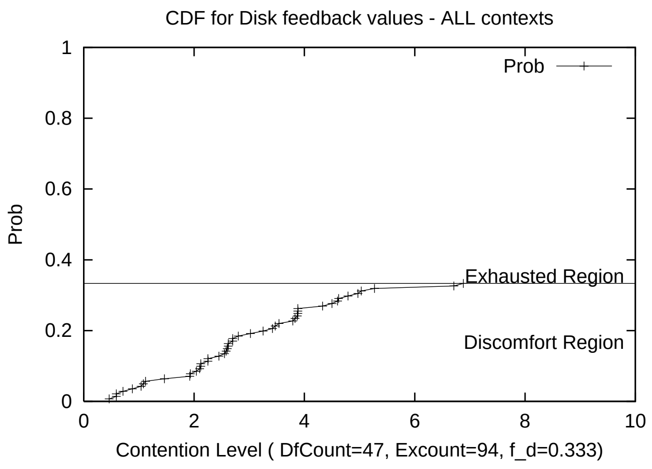


Figure 12. CDF of discomfort for Disk.

	CPU	Memory	Disk	Total
Word	L	L	L	L
Powerpoint	M	L	L	M
IE	M	M	H	M
Quake	H	M	M	H
Total	M	L	L	

Figure 13. User sensitivity by task and resource (Low, Medium, High).

gating over the four contexts, it appears we can easily borrow 33% of memory on these common PCs while irritating fewer than 5% of users ($c_{0.05,memory} \simeq 0.33$) in general.

Figure 12 shows the CDF for disk bandwidth. Almost 70% of users are comfortable even with seven competing tasks ($f_d = 0.33$). Furthermore, we can easily execute a single disk writing task, capable of consuming the whole disk bandwidth if run alone, while irritating fewer than 5% of the users ($c_{0.05,disk} \simeq 1.11$). We found this result remarkably counterintuitive as we ourselves tend to become uncomfortable when large amounts of unexplained disk I/O occurs on our desktops. The Dell machines we used for the study are remarkably quiet and have very dim disk lights. We suspect that it is the limited feedback about disk activity that leads to users accepting far higher amounts of disk contention than they otherwise might.

Note that the aggregated CDFs are intended to estimate performance over all applications. This is a rough estimate and will be more generalizable as we collect information for more applications.

	CPU	Memory	Disk
Word	0.71	0.00	0.10
Powerpoint	0.95	0.07	0.17
IE	0.75	0.30	0.61
Quake	0.95	0.45	0.29
Total	0.86	0.21	0.33

Figure 14. f_d by task and resource.

	CPU	Memory	Disk
Word	3.06	*	3.28
Powerpoint	1.00	0.64	3.84
IE	0.61	0.31	2.02
Quake	0.18	0.08	0.69
Total	0.35	0.33	1.11

Figure 15. $c_{0.05}$ by task and resource. (* indicates insufficient information)

3.3.2 How does the level depend on which resource or combination of resources is borrowed?

Figure 18 (located at the end of the paper) shows the CDF for each context and resource pair. Consulting its columns as well as the aggregated CDFs shown earlier clearly shows the strong dependence on the type of resource. Within the contention levels explored by the ramp testcases for each resource, users are much more tolerant with borrowing of memory and disk. This observation is qualitative as the testcases for each resource are different, but within the levels explored this holds true.

The varying tolerance by resource also shows up in our aggregated f_d, c_5 and c_a metrics, the column totals of Figures 14, 15, and 16. An important point to note is that the high f_d value of CPU (0.86), does not mean that the probability of discomforting users by borrowing CPU is 0.86. This probability depends on the contention. To determine this probability, a level must be chosen and the CDFs consulted as described in the previous section.

3.3.3 How does the level depend on the user's context?

In Figure 18, we see dramatic differences in the reactions to resource borrowing between different contexts. Consulting the rows illustrates this. It is clearly the case that the user's tolerance for resource borrowing depends not only on the resource, but also on what the user is doing.

The totals row of Figure 16 shows the average level at which discomfort occurs for the CPU contention for the four tasks. For an undemanding application like Word, the CPU contention can be very high (> 4) without significant affecting interactivity. However, with finer-grain interactivity, as in Powerpoint and Quake, the average level is much lower.

	CPU	Memory	Disk
Word	4.35 (3.97,4.72)	*	4.20 (1.89,6.51)
Powerpoint	1.17 (1.11,1.24)	0.64 (0.21,1.06)	4.65 (3.67,5.63)
IE	1.20 (1.07,1.33)	0.55 (0.39,0.71)	3.11 (2.69,3.52)
Quake	0.64 (0.58,0.69)	0.55 (0.37,0.74)	1.19 (0.86,1.52)
Total	1.47 (1.31,1.64)	0.58 (0.46,0.71)	2.97 (2.54,3.41)

Figure 16. c_a by task and resource, including 95% confidence intervals. (* indicates insufficient information)

This is likely due to the more aggressive CPU demands of these applications. Still, for the most aggressive application, Quake, results show that a thread with contention of nearly 0.2 can still run with a low probability of discomfort.

We used the same testcase for memory in all four tasks, growing the working set from zero to nearly the full memory size. The effect of memory borrowing is minimal in the case of Word (no discomfort recorded) and Powerpoint. IE and Quake are much more sensitive to memory borrowing, with more instances of discomfort ($f_d = 0.3$ and $f_d = 0.45$, respectively). For IE and Quake, value of $c_{0.05,mem}$ is 0.31 and 0.08 respectively, meaning that Quake users become discomforted at much lower levels. It appears that once office applications like Word and Powerpoint form their working set, significant portions of the remaining physical memory can be borrowed with marginal impact. This seems to be less true for IE and Quake, where their memory demands may be more dynamic.

Disk bandwidth can be borrowed with little discomfort in typical office applications. In Word and Powerpoint, the fraction of testcases ending in discomfort was small ($f_d = 0.01$ and $f_d = 0.17$ respectively), in the wide range covered by the testcases. IE and Quake are more sensitive. For identical disk testcases, we find that IE is more sensitive ($f_d = 0.61$). This may be expected as IE caches files and users were asked to save all the pages, resulting in more disk activity.

Figure 9 shows that users express feedback even when there is no testcase running. We note that users exhibit this behavior only in IE and Quake. Quake is a very demanding application in which jitter quickly discomforts users. There are sources of jitter on even an otherwise quiescent machine. Discomfort in IE depends to some extent on network behavior.

Figure 13 summarizes our judgement of user sensitivity to resource borrowing by resource and task. Note that the totals are not derived from the columns but represent overall judgements from the study of the CDFs (Figure 18).

App	Rsrc	Rating	p	Diff
Quake	CPU	PC Power vs. Typical	0.006	0.176
Quake	CPU	Windows Power vs. Typical	0.031	0.137
Quake	CPU	Quake Power vs. Typical	0.001	0.224
Quake	CPU	Quake Typical vs. Beginner	0.031	0.139
IE	Disk	Windows Power vs. Typical	0.004	1.114
IE	Mem	Windows Power vs. Typical	0.011	0.354

Figure 17. Significant differences based on user-perceived skill level.

3.3.4 How does the level depend on the user, factoring out context?

Users' comfort with resource borrowing depends to a small extent on their perceived skill level. We asked our users to rate themselves as $\{$Power User, Typical User, or Beginner$\}$ in each of $\{$PC Usage, Windows, Word, Powerpoint, IE, and Quake$\}$.

We compared the average discomfort contention levels for the different groups of users defined by their self-ratings for each context/resource combination using unpaired t-tests. In some cases, we found significant differences, summarized in Figure 17. The largest differences were for the combination Quake/CPU. For example, a Quake Power User will tolerate 0.224 less CPU contention than a Quake Typical User at a significance level (p-value) of 0.001. Even the users' self-rating for general Windows and PC use can lead to interesting differences in their tolerance. For example, for CPU, the differences between discomfort levels for Power and Typical users are quite drastic with $p = 0.002$ (PC Background) and $p = 0.010$ (Windows Background). Applications which have higher resource requirements show greater differences between user classes. However, our results are preliminary here and will improve with our Internet-wide study.

These results expose the psychological component to comfort with resource borrowing. Experienced or power users have higher expectations from the interactive application than beginners. When we borrow resources it may be helpful to ask the user to rate himself.

3.3.5 How does the level depend on the time dynamics of resource borrowing?

For this question, we have only preliminary results. We tested the "frog in the pot" hypothesis. A perhaps apocryphal rule in French cooking is that when boiling a frog, it is best to place the frog in the water before starting to heat it. The frog will not react to the slowly rising temperature, while a frog dumped unceremoniously into boiling water will immediately jump out. We paired ramp and step testcases in our study to explore if a similar phenomenon might be true of user comfort with resource borrowing—that a user would be more tolerant of a slow ramp that a quick step to the same level. We did observe the phenomenon in Powerpoint/CPU—the majority of users (96%) tolerated higher levels in the ramp testcase with a contention difference of 0.22 (averaged) with a p-value of 0.0001.

Our Internet-wide study is intended to address the question of time dynamics and of raw host speed more carefully.

4 Internet-wide study

Our controlled study at Northwestern helped us to answer or address many of the questions we raised in the introduction. We are now expanding both the scale and the questions through an Internet-wide study open to all participants. Any individual with a Windows computer is welcome to visit http://comfort.cs.northwestern.edu to download and run a copy of the UUCS client. The client is configurable by the user, including privacy options. We currently have about 100 users and are looking for many more.

In the Internet-wide study, the user uses the computer as normal. When user indicates any discomfort, the client records the context, the running processes, the contention levels, and other data similar to the controlled study. We plan to use this data to create better estimates for the aggregated resource CDFs (Figures 10-12), to better understand the effect of context and to measure the effect of the raw performance of the machine, which was not studied in our controlled study.

5 Advice to implementors

Based on our study, we can offer the following guidance to implementors of distributed computing and thin-client frameworks.

- Borrow disk and memory aggressively, CPU less so.
- Build a throttle [25]. Your system can benefit from being able to control its borrowing at a fine granularity similar to the UUCS client.
- Exploit our CDFs (Figures 10-12) to set the throttle according to the percentage of users you are willing to affect. As we collect more data, the CDF estimates will improve.
- Know what the user is doing. Their context greatly affects the right throttle setting.
- Consider using user feedback directly in your application.

6 Conclusions

We have described the design and implementation of a system for measuring user comfort with resource borrowing, as well as a carefully controlled study undertaken with the system. The end result has three components. First, we provided a set of empirical cumulative distribution functions that show how to trade off between the level of borrowing of CPU, memory, and disk resources and the probability of discomforting an end-user. Second, we describe how resource type, user context (task), and user-perceived expertise affect these CDFs. Finally, we have made initial observations on how the time dynamics of resource borrowing affect the level.

Surprisingly, disk and memory can be borrowed quite aggressively with little user reaction, while CPU can also be borrowed liberally. Our observations formed the basis of advice for the implementers of distributed computing and thin-client frameworks. We are currently exploring how to use user feedback directly in the scheduling of these frameworks and applications.

Acknowledgements

We thank Don Norman, Andrew Ortony, Ben Watson, Jack Tumblin, and Leslie Lamport for very useful suggestions. We also are grateful to the participants in our study.

References

[1] ANDERSON, T. E., CULLER, D. E., AND PATTERSON, D. A. A case for networks of workstations. *IEEE Micro* (February 1995).

[2] BHOLA, S., AND AHAMAD, M. Workload modeling for highly interactive applications. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (1999), pp. 210–211. Extended version as Technical Report GIT-CC-99-2, College of Computing, Georgia Tech.

[3] CHIEN, A. A., CALDER, B., ELBERT, S., AND BHATIA, K. Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing 63*, 5 (2003), 597–610.

[4] CURTIN, M., AND DOLSKE, J. A brute force search of des keyspace. *;login:* (May 1998).

[5] DINDA, P. A. The statistical properties of host load. *Scientific Programming 7*, 3,4 (1999). A version of this paper is also available as CMU Technical Report CMU-CS-TR-98-175. A much earlier version appears in LCR '98 and as CMU-CS-TR-98-143.

[6] DINDA, P. A., AND O'HALLARON, D. R. Realistic CPU workloads through host load trace playback. In *Proc. of 5th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR2000)* (May 2000). Springer LNCS 1915, pp. 265-280.

[7] DOUGLIS, F., AND OUSTERHOUT, J. Transparent process migration: Design alternatives and the Sprite approach. *Software Practice and Experience 21*, 7 (July 1991), 1–27.

[8] EMBLEY, D. W., AND NAGY, G. Behavioral aspects of text editors. *ACM Computing Surveys 13*, 1 (January 1981), 33–70.

[9] ENDO, Y., WANG, Z., CHEN, J. B., AND SELTZER, M. Using latency to evaluate interactive system performance. In *Proceedings of the 1996 Symposium on Operating Systems Design and Implementation* (1996).

[10] FIGUEIREDO, R., DINDA, P. A., AND FORTES, J. A case for grid computing on virtual machines. In *Proceedings of the 23rd IEEE Conference on Distributed Computing (ICDCS 2003* (May 2003), pp. 550–559.

[11] FREY, J., TANNENBAUM, T., FOSTER, I., LIVNY, M., AND TUECKE, S. Condor-g: A computation management agent for multi-institutional grids. In *Proceedings of the 10th International Symposium on High Performance Distributed Computing (HPDC 2001)* (2001), pp. 55–66.

[12] GOOGLE CORPORATION. Google compute. http://toolbar.google.com/dc/.

[13] GUPTA, A., LIN, B., AND DINDA, P. A Framework and Toolkit for Understanding User Comfort with Resource Borrowing. Tech. rep. NWU-CS-04-28, Department of Computer Science, Northwestern University, 2003.

[14] HUA CHU, Y., RAO, S., SHESHAN, S., AND ZHANG, H. Enabling conferencing applications on the internet using an overlay multicast architecture. In *Proceedings of ACM SIGCOMM 2001* (2001).

[15] JANNOTTI, J., GIFFORD, D., JOHNSON, K., KAASHOEK, M., AND JR., J. O. Overcast: Reliable multicasting with an overlay network. In *Proceedings of OSDI 2000* (October 2000).

[16] KLEIN, J. T. Computer response to user frustration. Master's thesis, Massachusetts Institute of Technology, 1999.

[17] KOMATSUBARA, A. Psychological upper and lower limits of system response time and user's preferance on skill level. In *Proceedings of the 7th International Conference on Human Computer Interaction (HCI International 97)*, pp. 829–832.

[18] LARSON, S. M., SNOW, C. D., SHIRTS, M., AND PANDE, V. S. Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology. In *Computational Genomics*, R. Grant, Ed. Horizon Press, 2002.

[19] LITZKOW, M., LIVNY, M., AND MUTKA, M. W. Condor — a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems (ICDCS '88)*, pp. 104–111.

[20] MUTKA, M. W., AND LIVNY, M. The available capacity of a privately owned workstation environment. *Performance Evaluation 12*, 4 (July 1991), 269–284.

[21] REYNOLDS, C. J. The sensing and measurement of frustration with computers. Master's thesis, Massachusetts Institute of Technology Media Laboratory, 2001.

[22] RIPEANU, M., FOSTER, I., AND IAMNITCHI, A. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal 6*, 1 (2002).

[23] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)* (2001).

[24] RYU, K. D., AND HOLLINGSWORTH, J. K. Fine-grain cycle stealing for networks of workstations. In *Proceedings of ACM/IEEE SC98 (Supercomputing '98)* (November 1998), pp. 801–821.

[25] RYU, K. D., HOLLINGSWORTH, J. K., AND KELEHER, P. J. Efficient network and I/O throttling for fine-grain cycle stealing. In *Proceedings of Supercomputing '01* (November 2001).

[26] SAPUNTZAKIS, C., BRUMLEY, D., CHANDRA, R., ZELDOVICH, N., CHOW, J., LAM, M. S., AND ROSENBLUM, M. Virtual appliances for deploying and maintaining software. In *Proceedings of the 17th Large Installation Systems Administration Conference (LISA 2003)* (October 2003).

[27] SCHMIDT, B., LAM, M., AND NORTHCUTT, J. The interactive performance of slim: A stateless thin client architecture. In *Proceedigns of the 17th ACM Symposium on Operating Systems Principles (SOSP 1999)* (December 1999).

[28] SHARMAN NETWORKS. The Kazaa Media Desktop. http://www.kazaa.com.

[29] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, F., AND BALAKRISHNAN, H. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM 2001* (2001), pp. 149–160.

[30] SULLIVAN, W. T., WERTHIMER, D., BOWYER, S., COBB, J., GEDYE, D., AND ANDERSON, D. A new major seti project based on project serendip data and 100,000 personal computers. In *Proceedings of the Fifth International Conference on Bioastronomy* (1997).

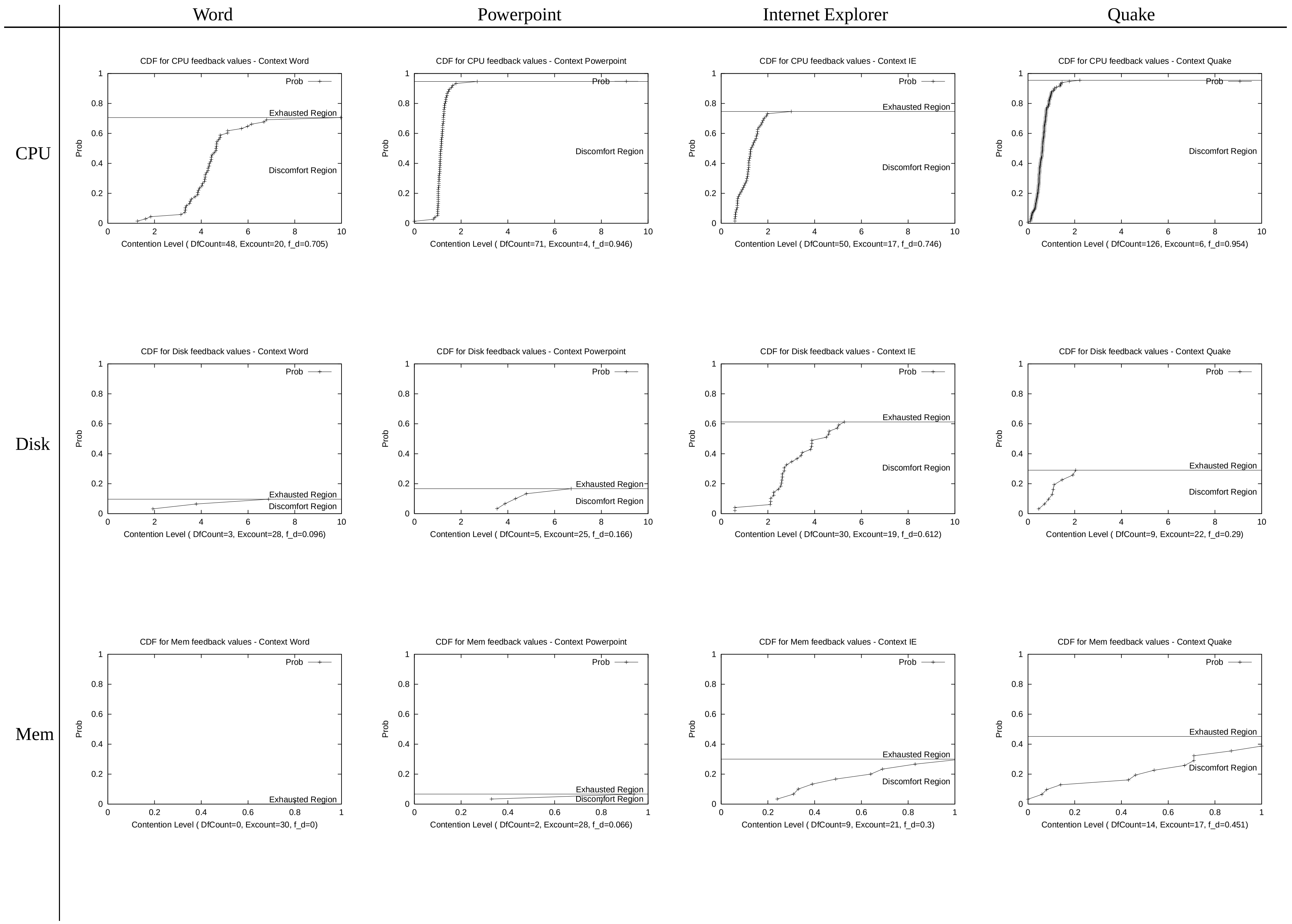


Figure 18. CDFs by resource and task.