Workload Propagation – Overload in Bursty Servers

Qi Zhang
Department of Computer Science
College of William and Mary
Williamsburg, VA 23187
qizhang@cs.wm.edu

Alma Riska
Seagate Research
1251 Waterfront Place
Pittsburgh, PA 15222
alma.riska@seagate.com

Erik Riedel
Seagate Research
1251 Waterfront Place
Pittsburgh, PA 15222
erik.riedel@seagate.com

***Abstract*—Internet servers are developing into complex but central components in the information infrastructure and are accessed by an ever-increasing and diversified user population. As such, they are susceptible to unpredicted and transient overloads, making highly available, yet cost-effective service a critical challenge. System responsiveness during overload periods is at least as important as system behavior under average, steady state load. Although underutilized in normal operation, the system can degrade to the point that user-visible performance is impacted, often severely, when subject to overload bursts. Using measurements from a 3-tier e-commerce server, we show how normal and overload conditions propagate through the system hierarchy, i.e., from the front-end server to the back-end database, focusing on CPU, memory, and I/O performance. In particular, we focus on how system workload is transformed by the time it reaches the lowest system level, i.e., the disk drives storing the permanent database. We find that overload conditions at lower system levels occur not only when the *number* of users in the system increases, but also during rapid changes in the *nature* of the work requested by the current users. These observation advocate the development of system mechanisms at the lower levels that can detect overload and self-adapt their configuration parameters in order to ensure high service availability and graceful performance degradation. We illustrate the effectiveness of a scheduling mechanism at the disk and provide a proof of concept that such self-adaptive scheduling mechanisms at the lower levels are a step toward faster system recovery under conditions of transient overload, and can thus complement admission control at the front end.**

Keywords: workload characterization, multi-tiered systems, overload conditions, TPC-W.

I. Introduction

In a clear shift from the early Web days where all content in Internet servers was static files, contemporary servers provide mostly content that is dynamically generated. Resource provisioning in systems that handle only static requests is straightforward. For static requests, it has been shown that caching is effective and that service times are linear to the transfered file sizes. Correspondingly, the system bottleneck is the limited network bandwidth [9]. Understanding the resource requirements of dynamic requests is more challenging. It is possible that a request will cause a substantial portion of the database to be accessed even when just a few kilobytes of text are eventually sent back to the client. The "size" of a request is not a simple one-dimensional property, and thus is difficult to quantify a priori. For a dynamic request, the amount of data accessed from storage, the computational requirements for processing in order to generate the page content, and the amount of data sent back to the client are unrelated and unpredictable. Furthermore, wide disparities in the various resource requirements of different dynamic requests, trigger multiple bottlenecks in the system.

This work was partially supported by the National Science Foundation under grants ITR-0428330 and CCR-0098278 and ACI-0090221.

Resource allocation is further complicated by bursty user request rates that fluctuate dramatically even within short periods of time, resulting in systems that operate often under conditions of transient overload.

Effective system provisioning in such complex systems requires a detailed understanding of the system workload. While over provisioning helps Internet systems to operate under light to medium load conditions, it does not preclude the fact that they are still susceptible to transient overload. At the beginning of an overload period, the system starts operating in a "yellow zone" where response times increase, and all users (or selected classes of users) [27] experience performance degradation. If overload conditions persist, then the system starts operating in a "red zone", resulting in response times so slow that are equivalent to service unavailability. Overload can be controlled via admission control mechanisms either at the front-end of the system (usually the network link) or at the application level, with the intension of stabilizing performance [5], [15], or via effective resource management that focuses on graceful performance degradation [27]. Performance degradation is by far more desirable than service unavailability. The latter, even if it occurs for a short time period only, can be detrimental for business profitability [14].

The purpose of this paper is to present a detailed analysis of the resource demands in a typical e-commerce server under *steady load* and under *transient overload*, to identify how the workload propagates through all system tiers, and to determine the conditions under which bottlenecks occur. Typical configurations of an e-commerce site use a 3-tier architecture consisting of a web and application server, a database server, and a storage system (see Figure 1). We have built an experimental system of an on-line bookstore using the TPC-W workload [22], the current industry standard e-commerce benchmark, and have conducted measurements at all three tiers of the system. We are particularly interested in how the workload's transient characteristics propagate through the system hierarchy and place resource demands on CPU, memory, and I/O devices. Being able to detect overload states as well as identify the conditions under which bottlenecks arise or switch among devices or tiers is the first step for designing effective admission control mechanisms at the front-end and/or effective resource allocation policies in each device. Faster recovery from the "yellow" operating zone to normal operating conditions, and most importantly avoiding (or delaying) entering the "red" zone where the system has no other choice but rejecting service, are critical for the commercial success of any Internet server.

This paper is organized as follows. Related work is presented in Section II. Section III gives an overview of the experimental set up and the tracing mechanisms we use across the various tiers. In Section IV, we present performance results and identify systems bottlenecks under *steady state* conditions, i.e., when the system is loaded (lightly or heavily) but load remains steady throughout the experiment. In Section V, we show how overload develops and propagates in the system under *transient conditions* that are distinguished by sudden changes in the customer arrival intensity and/or changes in the requested work. Section VI gives an overview of what we have learned from these experiments and presents a case study that shows that overwork can be detected (and handled via self-adaptive policies) at the lowest tier, most specifically the storage system. Finally, Section VII summarizes our contributions and outlines future work.

II. Related work

Internet servers and services have evolved from centralized and information based only to personalized e-commerce, distributed peer-to-peer, global storage, and grid-based. Early studies pointed out the existence of burstiness in Internet-related systems [19], [12]. Recent studies show that burstiness persists [27], [5], but better understanding is required for the more complex and sophisticated nature of both Internet services and systems, which is a direct result of the personalized nature of Internet-related services that need an array of resources, i.e., CPU, memory, and I/O, for serving requests [3], [17], [18]. The complexity of the architecture of Internet systems, the ever-increasing set of services that they provide, and the highly diversified user population makes such systems susceptible to transient overload.

Data on actual e-commerce sites are difficult to obtain as they are subject to non-disclosure agreements. Consequently, one can only resort to synthetic workload generators to study such systems, the most prominent being the TPC-W benchmark for an e-commerce site [22]. Studies based on TPC focus on bottleneck identification and find a variety of causes [8], [2], [26], [13]. Note that the TPC benchmarks implement stationary arrivals only, which gives a restricted view of the system: observing the system in steady state does not tell us how the systems behaves in transient overload.

In contrast to the above works, we concentrate on how load propagates deep into the tiers of an e-commerce system, and examine how transient behavior affects bottlenecks and workload propagation. Via experimentation we demonstrate that overload can be caused by *both* extreme load (i.e., number of users) and extreme work conditions (i.e., type of requests) and therefore it can appear unexpectedly at various levels of the system hierarchy, making a case for the need of techniques to detect and handle overload/overwork across the entire multi-tiered system.

Currently, QoS mechanisms usually target multi-class systems and achieve performance differentiation by prioritizing scheduling of system resources at the front-end tier (i.e., at the application server or web server) [10], [23], [11]. Even for systems that do not explicitly provide QoS guarantees, it is critical to achieve graceful degradation of performance in case of overload. Several alternatives have been proposed to achieve graceful performance degradation at the front-end [25], [5], [4], some are based on a control-theoretic approach [1], while others use monitoring of end-to-end performance to trigger admission control [6]. A service design framework that detects and handles overload at *all* system levels is presented in [27] and is based on establishing a feedback loop between the application and system, requiring application changes in order to provide proper response to overload.

We believe that the development of QoS policies at the lower tiers which adapt their configuration parameters according to the current demands is a good complementary to the front-end admission control. In this paper, we proposed an overload-aware disk scheduling algorithm in the storage system handling overload in this tier.

III. Experimental environment

We evaluate workload propagation in a multi-tier server via measurements in an e-commerce site that simulates the operation of an on-line bookstore, according to the TPC-W benchmark [22]. A high-level overview of the experimental set-up is illustrated in Figure 1 and specifics of the software/hardware used are given in Table I.

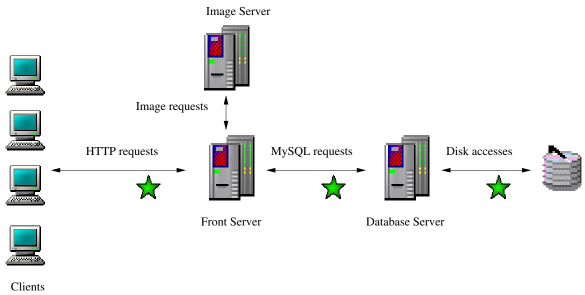


Fig. 1. E-commerce experimental environment.

Because we focus on the activity across all tiers in the system, we collect measurements at the front-end server (that hosts the web and application servers), the back-end database server, and the disk. In Figure 1, all measurement points are denoted with a star (*). Trace data are collected via the following utilities:

- Arrivals at the front-end server are obtained by tracing the workload generation modules.
- CPU and memory activity at each server is measured via the *sysstat* Linux utility.
- Query activity at the database server is provided by MySQL logs.
- VMware [24] is used to run the database server in a Linux virtual machine hosted by the database server machine. This allows the physical SCSI disk to appear as a process in the database host. We use the *strace* Linux utility to trace all I/O activity.

We separate the image files from the front-end server in an effort to minimize their effect. The host of the database server has 1 GB of memory but the virtual machine uses only 768 MB. We deliberately selected a small system so that it could be easily measured and understood. Our focus is on the conditions that overload happens, so a small system is easier to overload than a

	Processor	Memory	OS
Clients (Emulated-Browsers) [20]	Pentium 4 / 2 GHz	256 MB	Linux Redhat 9.0
Web Server - Apache2.0/Tomcat4.0 [21]	Pentium III / 1.3GHz	2 GB	Linux Redhat 9.0
Database Server - MySQL4.0 [16]	Intel Xeon / 1.5 GHz	1GB / 768 MB	Linux Redhat 9.0
Disk	SEAGATE: ST373453LC; SCSI; 73 GB; 15,000 rpm		

TABLE I

HARDWARE COMPONENTS ON THE ON-LINE BOOKSTORE IMPLEMENTATION

large one. Furthermore, a system with 1 GB of memory for only a single disk is clearly over provisioned and thus optimistic, a real system would have a much larger number of disks. In the following sections we show that the database memory can become the bottleneck device depending on the type of transaction requests.

The database of the online store has 10 tables. One of the most important ones is the ITEM table which stores information on the items available for purchase. The database size is determined by the number of items and the number of customers. In our experiments, we found that the size of the ITEM table is critical for performance. Therefore, we present results on different databases that are distinguished by the size of the ITEM table. We run experiments in three databases: one with 10,000 items (small), one with 100,000 items (medium), and one with 1,000,000 items (large). Table 2 shows the size of the most important tables in the databases used in our experiments.

TPC-W defines 14 different Web interactions which are coarsely classified as either browsing or ordering. According to the weight of each type of activities in a given traffic mix, TPC-W defines 3 types of traffic mixes, namely, the *browsing mix* with 95% browsing and 5% ordering, the *shopping mix* with 80% browsing and 20% ordering, and the *ordering mix* with 50% browsing and 50% ordering. Because our focus is on workload propagation across all tiers under dynamic traffic requirements, we conduct experiments using first the default traffic mixes as defined by TPC-W and then specifically modified variants as explained in detail in the following sections. We stress that the reason for experimenting with different workload mixes is to show relative changes in response times, not the absolutes a "real" user would see.

DB	ITEM	CUSTOMER	ORDER_LINE	Total
10 K	5.1 MB	362 MB	338 MB	1.5 GB
100 K	51 MB	362 MB	338 MB	1.5 GB
1 M	510 MB	362 MB	338 MB	2.1 GB

TABLE II

SIZES OF IMPORTANT TABLES

IV. STEADY STATE WORKLOAD

Recall that TPC-W is a model of a *closed* system, i.e., it simulates the behavior of a system given that the number of customers in the system is fixed for the entire experiment. This essentially imposes a steady load (i.e., arrival intensity) into the system. In this set of experiments, we seek to identify system capacity under steady state conditions. From one experiment to the next, we increase the system load by increasing the number of EBs, until overload. The purpose is to find the number of browsers that "stresses" the system such that bottlenecks (or switching of bottlenecks) are identified. Each experiment lasts 30 minutes and we discard all data from the first 15 minutes to mask out warm up effects. We monitor averages of overall system throughput, defined as the number of replies received by the EBs per second, and memory and CPU utilizations at the front-end and database servers. Table III presents the bottleneck and the number of browsers that overloads the system for the three traffic mixes as defined by TPC-W (i.e., browsing mix, shopping mix and ordering mix) and three databases sizes as defined in Table II.

Observe that the system throughput of the small database is consistently higher than the other two databases. This is because the costlier database transactions are the browsing requests, which do "search" and "sort" operations but commit no changes. For instance, the "New Products" transaction examines and sorts 400 records in the 10K items database, 4,000 records in the 100K database, and 40,000 records in the 1M database. Hence, the database server CPU is the system bottleneck for the small and medium databases, while for the large database the bottleneck moves to the database server memory, where memory thrashing occurs because of the frequent sequential accesses to the 510 MB ITEM table.

The effect of changes in the nature of the work done by the customers is illustrated by the different capacities under the three workload mixes. Less browsing and more ordering results in faster system response time. Therefore under the shopping and ordering mixes, the system sustains larger capacities than under the browsing mix. However shopping mix and browsing mix generate the same bottlenecks in the system under heavy load, i.e., the database server CPU for the small and medium databases and the database server memory for the large database. While the ordering mix under heavy load switches the system bottleneck to the front-end web server for the smaller databases, but still has the database server memory as the bottleneck for the largest database. [1].

We summarize our observations for TPC-W under steady state conditions as follows:

- Different TPC-W mixes sustain different arrival intensities (i.e., *load*), as this is expressed by number of EBs.
- The type of *work* done by the EBs is critical for bottleneck identification, and the point where bottleneck occurs.

[1] Note that we report results up to 640 EBs as the front-end server could not support more EBs due to Java and Linux limitations. Extrapolating from the figure trends, we observe that, the front-end server CPU becomes the bottleneck for the 10K and 100K databases.

	10K items		100K items		1M items	
	Bottleneck	Capacity	Bottleneck	Capacity	Bottleneck	Capacity
Browsing Mix	DB CPU	256	DB CPU	192	DB memory	80
Shopping Mix	DB CPU	480	DB CPU	320	DB memory	96
Ordering Mix	Front CPU	> 640	Front CPU	> 640	DB memory	192

TABLE III

SYSTEM BOTTLENECK AND CAPACITY IN TERMS OF NUMBER OF EBS UNDER DIFFERENT WORKLOADS

- Switching of bottlenecks does happen depending on the work being done by the browsers (i.e., the TPC-W mix).
- The database size does have an important effect on determining the system bottleneck.

In the following section, we elaborate on how changes in both *load* and nature of *work* can affect system performance.

V. TRANSIENT WORKLOAD

After having understood what the system bottlenecks are under steady state conditions, we now turn to transient analysis as systems in the real world are rarely subject to such stability in their workload. For the specific case of the e-commerce server, a popular new product or a seasonal sale could lead to sudden increases in arrival intensity. Different arrival patterns may be experienced during day or night times. Furthermore, the pattern of user requests, i.e., the information they request from may be substantially different from the expected one, affecting the demands put on system resources. In this section, first, we concentrate on how ephemeral changes in the arrival intensity (i.e., load) propagate through the system tiers and affect system performance. Then, we focus on how changes in the type of requests (i.e., work) affect the bottleneck resource and how these patterns are inherited into lower system levels. For all experiments, we present activity in all three tiers across time. All experiments are run for 60 minutes. Results of the first 20 minutes are ignored to mask out warm up effects.

Experiment One (10K Database, Transient Load): The first experiment uses the browsing mix of TPC-W but changes the number of active browsers in a controlled manner. For the experiment that uses the small database (see left column of Figure 2), for the first 600 seconds the number of browsers is set to 64, for the next 300 seconds it is set to 384 to induce a short-lived overload condition, and for the remaining 25 minutes of the experiment the number of EBs is reset to 64. Figures 2(I.a)-2(I.c) show the intensity of arrivals across time at the front-end server, database server, and the database disk, respectively. Note the flux of arrivals from one server to the other, as well as the time where significant increases/decreases are shown. The transient load causes a severe increase in the arrival intensity at all tiers except the disk, where short arrival bursts are detected over time with an exception toward the end of the overload period (after the 800^{th} second) where the arrival burst is sustained longer.

Figure 2(I.d) reports on the disk access pattern as a function of time. The entire disk is mapped on the y-axis which marks the physical layout of each database table on the disk. Vertical lines in this figure indicate disk sequential accesses that correspond to an entire table being scanned. Clearly, there is higher intensity of disk activity during the period of the arrival burst, which makes disk transfers slower, especially after the 800^{th} second. Note that entire table accesses are not shown as vertical lines anymore, indicating that these accesses take now longer. The effect of the bursty arrivals on the query response time are reported in Figure 2(I.e). As expected, during the overload period database queries take longer to complete which negatively affects user perceived performance.

Note that since TPC-W simulates a closed system, the arrival rate to the first server is an equivalent indicator of system throughput. Observe that during the overload condition both arrival intensity and system throughput increase (Figures 2(I.a) and 2(I.b)). This overload propagates from the front-end web server down to the database server, but not to the disk. System performance is affected (Figure 2(I.e)), but not to the point of affecting system throughput. We conclude that the system sustains its performance during the short-lived overload period.

Experiment Two (1M Database, Transient Load): The right column of Figure 2 reports on system performance at various levels for the large database and the browsing mix. Recall that for the experiments with the large database, memory may become the bottleneck as the ITEM table is 512 MB and the available memory only 768 MB. In this experiment, for the first 600 seconds the number of browsers is set to 16, for the next 300 seconds it is set to 96 to cause a short-lived overload condition, and for the remaining 25 minutes is reset to 16. Note that the arrival intensity propagates from one tier to the next, reaching the database disk finally. Yet, there is a significant drop in the arrival rate/throughput at the front server after the 900^{th} second despite the fact that the bursty period ended and the number of EBs is restored to 16, see Figure 2(II.a). This drop persists for the next 400 seconds and it is reflected on the database server also, see Figure 2(II.b).

The antipode is observed during the same time period at the disk, see Figure 2(II.c). High arrival rates at the disk imply the presence of severe queueing at the storage system, resulting in slower response times. Because the TPC-W simulates a closed system (i.e., the number of EBs in the system remains the same during the experiment), the majority of the requests are accumulated in the slowest tier, resulting in a significant drop in the rate of request completions, which reduces the throughput and consequently arrival rates to the front and database servers (see Figures 2(II.a)–2(II.b)). We emphasize that the drop of the throughput at the front-end and at the database server is as drastic as to imply conditions of service unavailability for the period of time that the storage system suffers from overload.

Figure 2(II.d) further sheds light to this behavior. After the 800^{th} second the system clearly operates under conditions of severe overload. The system recovers after the 1300^{th} second, well after the burst ends and the number of browsers is reset to 16. This overload period at the disk is the result of memory thrashing at the database server, causing repetitive, long sequential scans to the ITEM table. Under normal conditions the ITEM table is fully stored in memory. In this experiment, the sequential scans of the ITEM table are represented as *almost* horizontal lines in Figure 2(II.d) indicating that it takes a long time to complete some of the database queries. This is also reflected in the large gap between the query response time distributions during normal and overload conditions in Figure 2(II.e). Approximately 5% of all queries have a response time of more than 1 minute, which indicates service unavailability.

Concluding on experiments one and two, we stress that it is important for performance that overload propagates up to the database server CPU only (experiment one, Figure 2(I)) rather than further down to the database disk (experiment two, Figure 2(II)). In experiment one, overload does not affect system throughput and the system recovers as soon as the bursty period ends. In experiment two system throughput drops to the point of service unavailability and the recovery process is much longer than the bursty period itself. Overload propagation down to the storage system slows down the entire system operation, and significantly affects system availability.

The results of the first two transient load experiments highlight a case that can be managed via admission control at the front-end. However, it is possible that overload happens when the number of arrivals remains the same at the upper system tiers: all that is needed is to change the type of work requested by the customers. For this new set of experiments that simulate transient work in the system, we fix the number of customers but we change the nature of their work for 300 seconds (from the 600^{th} until the 900^{th} second).

Experiment Three (10K Database, Transient Work): We report results for the small database using the ordering mix with 640 EBs on the left column in Figure 3. Recall that under the ordering mix the front-end rather than the database server operates close to its capacity and 640 EBs do not saturate system resources in the lower tiers. In this experiment, we keep the number of EBs steady but from the 600^{th} to the 900^{th} seconds of the experiment, we change the work done by the browsers as follows: we increase the percentage of ORDER_DISPLAY requests from 0.22% to 30% and we proportionally adjust the percentage of the rest of the requests in the ordering mix. ORDER_DISPLAY requests search in the CUSTOMER, ADDRESS, and ORDER_LINE tables to generate reports on all orders placed by a single customer and on related best-selling items for each order. Note that we do not introduce any new query in the TPC-W workload, we only modify the weight that each of the pre-defined transactions have to simulate an unusual workload mix in the system

Figures 3(I.c)–3(I.d) confirm that there is significantly higher disk activity in multiple tables, in contrast to the disk activity of the browsing mix in experiment one (see Figure 2(I.d)), indicating that the database working set is substantially larger under the ordering mix and its variant than for the browsing mix. Recall that the size of the entire small database is approximately 1.5 GB, suggesting that a large working set would not fit into the database server memory of 768 GB. This is the reason that the disk becomes the bottleneck with the ordering mix variant. As a result, arrival rates (and consequently system throughput) at the front-end and database servers significantly drop from the 600^{th} until the 950^{th} second (see Figures 3(I.a) and 3(I.b)), suggesting that the system operates in overload. System throughput is reduced during the period of bursty work, but does not become zero as in experiment two. Overwork in experiment three, similarly to the overload in experiment one, impacts negatively the user-perceived performance (see Figure 3(I.e)) but does not drive the system to unavailability. Experiment three shows that extreme work conditions slow down the lowest level of the system, i.e., the database disk, even when system resources are well-provisioned. Recall that the sizes of important tables in the small database (see Table 2) allow the database working set to fit comfortably in the 768 MB memory. Comparing results of experiments one and three, one can see that while load did not propagate down to the lowest tier, work did propagate. This indicates a critical difference between overload and overwork: work can propagates further down to the database disk and can makes the system susceptible to overload independently of the available system resources.

Experiment Four (1M Database, Transient Work): The last experiment reports on performance data using the large database and the browsing mix and is illustrated on the right column of Figure 3. Now the number of browsers is set to 32 throughout the experiment. In the TPC-W browsing mix, the percentage of requests that ask for new products is 11%. For 5 minutes (from the 600^{th} till the 900^{th} second), 90% of requests are for new products, with the remaining of the requests proportionally adjusted. After the 900^{th} second, the system operates under the default browsing mix again. This change drastically increases the arrival rate to the database disk, as the ITEM and AUTHOR tables need to be accessed (see Figures 3(II.c)–3(II.d)). These disk accesses significantly increase the average query response time during the sensitive period (see Figure 3(II.e)). During the bursty work period, throughput at the front-end web server reduces to zero for nearly 200 seconds beyond the end of the bursty period at the 900^{th} second (see Figures 3(II.a)). This behavior, similar to experiment two, indicates system unavailability during the transient overwork period.

Recall that the system with the large database has less available resources than the system with the small database as its working set is larger but the available system memory is the same. Under both transient load (experiment two) and transient work (experiment four), the system becomes unavailable. For the experiments with the small database (experiments one and two) the system resources are well provisioned, as a result during overload or overwork performance drops but the system quickly recovers and does not become unavailable.

In contrast to the transient load which may be detected by simply monitoring the arrival rate at the system front-end, transient work may be detected by observing inorderly slower response times, despite the fact that the number of users in the system (or the number of connections) remains relatively unchanged. Yet, we showed that even for the small database where steady state analysis (see Section IV) shows that memory is hardly the bottleneck, scenarios that make the system suffer from severe overload are not hard to devise. We summarize our observations for the cases of transient load and transient work experiments as follows:

- If overload due to transient load or transient work propagates down to the lowest tier, i.e., the database disk, the effect is reflected in the system hierarchy up to the highest tier, as dips in the arrival rate/ throughput in the front end indicate that the system shows signs of service unavailability.
- Effective system provisioning is difficult, as changes in the work done may quickly drive system resources to saturation, and system recovery from these states may become very slow.

In the next section we elaborate on methods to speed up system recovery, focusing on the resource allocation policies at the storage system.

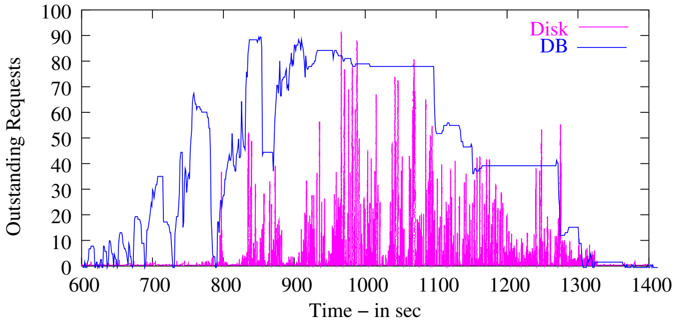
VI. System Implications

In the previous sections, we showed via measurements how the system workload propagates down the tiers of an e-commerce site. Our goal is to understand the conditions under which certain tiers of the system become the bottleneck and negatively affect service availability and user-perceived performance. The straightforward conclusion is that too much load at the front-end generates too much load at the lower levels of the system, increasing the average request service time and causing the service to become unavailable. Our experiments also showed that excessive work, especially in the lower tiers, may also cause response times to become so slow that can considerably degrade user-perceived performance. By propagating down the system hierarchy, excessive work critically affects system performance and, similarly to excessive load, might bring system availability to a halt as depicted in Figures 2(II.a)–(II.b) and 3(II.a)–(II.b.) Consequently, effective handling of such overload conditions, e.g., via *work-shedding* of some form, becomes as important as any *load-shedding* technique.

The straightforward way to avoid overloading is admission control, i.e., reject service to new customers at the system front-end and/or interrupt service to existing users. Note that from the perspective of the service provider, service interruption to existing users bears more penalty than rejection of service to new users. As shown in Section V, excessive work might result even from the same set of users, which leaves service interruption at the front-end as the main alternative to sustain service availability during the transient overwork period.

We believe that complementary to front-end admission control is the development of work-shedding policies at the lower tiers of the system that adapt their configuration parameters according to the current resource demands. In fact, it is possible for system resources to achieve much of the benefits of work-shedding by taking advantage of local information and by understanding their own behavior. In contrast to overload which appears both at the the front-end and the lower tiers, overwork only appears at one tier, i.e., the bottleneck tier. Figure 4 illustrates the number of outstanding requests (i.e., the queue length) at the database server and the storage system during the overload and overwork scenarios in experiments two and four of the previous section. In the overload case (see Figure 4(a)), queue lengths of both resources follow the same pattern, while for the overwork case (see Figure 4(b)) spikes in the queue length at the disk and the database are complementary to each other across time.

(a) 1M database under transient (over)load; Experiment two



(b) 1M database under transient (over)work; Experiment four

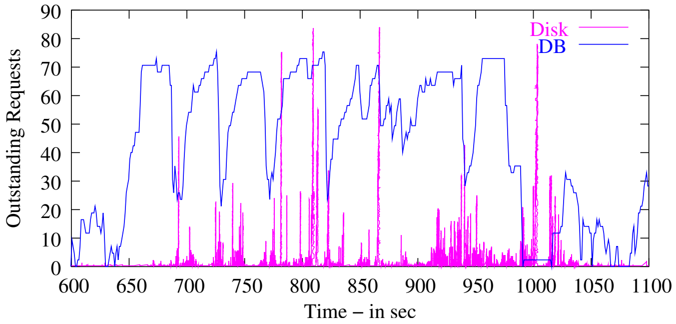


Fig. 4. Outstanding requests (queue lengths) at the disk and at the database during overload and overwork scenarios of (a) experiment two and (b) experiment four.

This further shows that lower tiers may have an advantage in work-shedding over the front-end. The front-end cannot easily distinguish the *source* of slow response time: it is because of increased arrival intensity (overload), or is it because of a slower service process (overwork)? Furthermore, the front-end cannot make the best decisions about which requests to drop, as it cannot accurately assess which are expensive and which are not. As a result, it may end up dropping requests randomly, leaving a large fraction unserviced [27] instead of a small fraction of well-chosen requests. For requests that have already reached deep in the system hierarchy and cannot be easily rejected, simple work reordering, i.e., preferential scheduling in the same spirit as in [9], may result in faster system recovery from overload. For example, the system resource may choose to postpone certain actions in order to achieve graceful degradation in system performance and avoid service unavailability. This can be an effective strategy especially if the overload (or overwork) condition is only temporary. In the next subsection, we give a proof-of-concept that it is possible for system resources to make independent decisions toward effective overload/overwork handling.

A. Case study: handling overload/overwork at the storage system

Now, we focus only at the lowest level in the system, i.e., the storage system, and propose a technique to handle disk overload/overwork. Our goal is to adapt disk operation such that during overload, or overwork, graceful performance degradation is achieved. In order to handle overload at the disk where service rejection is not straightforward, we prioritize the work such that the tail of disk response time, i.e., the worst case, is contained.

Note that a single request for a database table gets transformed to several, sometimes hundreds of, I/O requests. If these I/O requests are for consecutive blocks of the disk media, they are considered to be a sequential stream. A fully random I/O workload instead consists of requests uniformly distributed over the disk media. In an e-commerce system, usually, the disk workload consists of a mix of sequential streams and random requests, as Figures 2(I.d), 2(II.d), 3(I.d), and 3(II.d) illustrate. By characterizing the I/O workload as a set of streams, one can identify long running streams and short running streams. This characterization becomes very useful in times of overload where it may be better to prioritize service of short-running streams and postpone service of the long-running ones, in the same spirit as the Shortest Job First (SJF) family of policies. We implemented this prioritization of streams into the Shortest Positioning Time First (SPTF) policy [28], which is widely implemented in disk drives today. We detect overload/overwork at the disk by monitoring its queue length. Once queue length reaches a predefined threshold, SPTF serves short-running streams and postpones the long-running ones for the future. The postponed requests are served after the transient overload period ends or a predetermined time interval has elapsed.

By viewing the I/O workload as streams of requests, decisions at the disk level can affect only a small number of database requests.

We evaluate this disk overload-aware policy via trace driven simulation. We use DiskSim [7] as the disk-level simulator, which we drive using traces from the testbed described in Section III, specifically traces from experiments two and four (see Figure 2(II) and Figure 3(II)). Here, we concentrate on handling overload at the disk level only. A comprehensive evaluation of the performance implications of this disk scheduling policy in the entire system is the subject of future work.

Figures 5 and 6 present the response times of all disk requests during the overload and overwork periods in experiments two and four, respectively. Results are presented as a function of time using the normal SPTF scheduling policy and the overload-aware SPTF. Observe that for both experiments, disk response times using the overload-aware scheduling algorithm are at least twice as low. This fact is emphasized also in the complementary distribution functions of the response times for both policies (see Figures 5(c) and 6(c)), which show that there is a clear reduction in the tail of the response time distribution when overload-aware scheduling is used. This reduction can greatly help in avoiding the time during which the system operates on a "red" zone, i.e., when the system is so slow that it is unavailable. By modifying the disk scheduling discipline, we reduced the worst response times at the disk by a factor of two, clearly improving the responsiveness of the lowest tier and improving on the overall system availability.

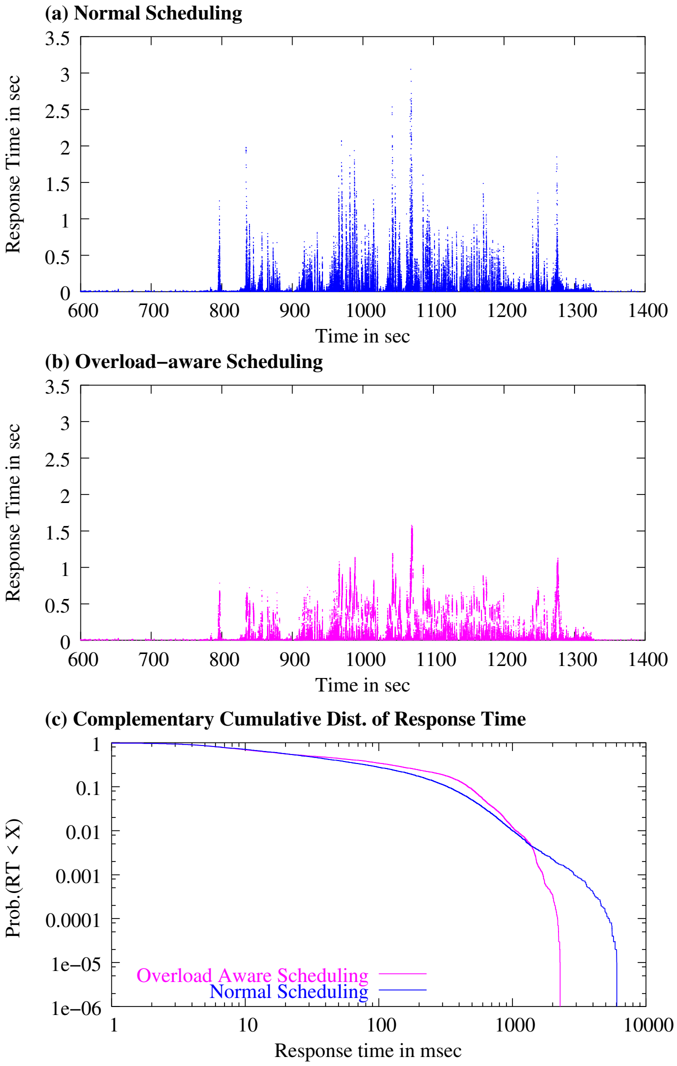


Fig. 5. Response time at the disk as a function of time under (a) normal scheduling, (b) overload-aware scheduling, and (c) the complementary cumulative distribution during the overload period of experiment two.

VII. Conclusions

We have done a detailed workload characterization study via experimental measurements in a 3-tier e-commerce system built according to the TPC-W specifications to evaluate how workload propagates through all levels of the system hierarchy. More specifically, we studied the performance effects of transient excessive load in the system (i.e., an higher than usual number of users, number of web sessions, or number of network flows) as well as the effects of transient excessive work (i.e., sudden increase in the demand of system resources by the current users of the system).

We measured resource utilization through all 3 tiers of the system, i.e., at the front end web server, at the database server, and at the database disk and showed that it is the lower tiers that suffer most from such overload/overwork conditions. The further overload/overwork propagates down the system hierarchy, i.e., the memory/disk, the higher the performance penalty, and the more difficult it is to handle it effectively. Complementary to front-end admission control mechanisms, effective re-

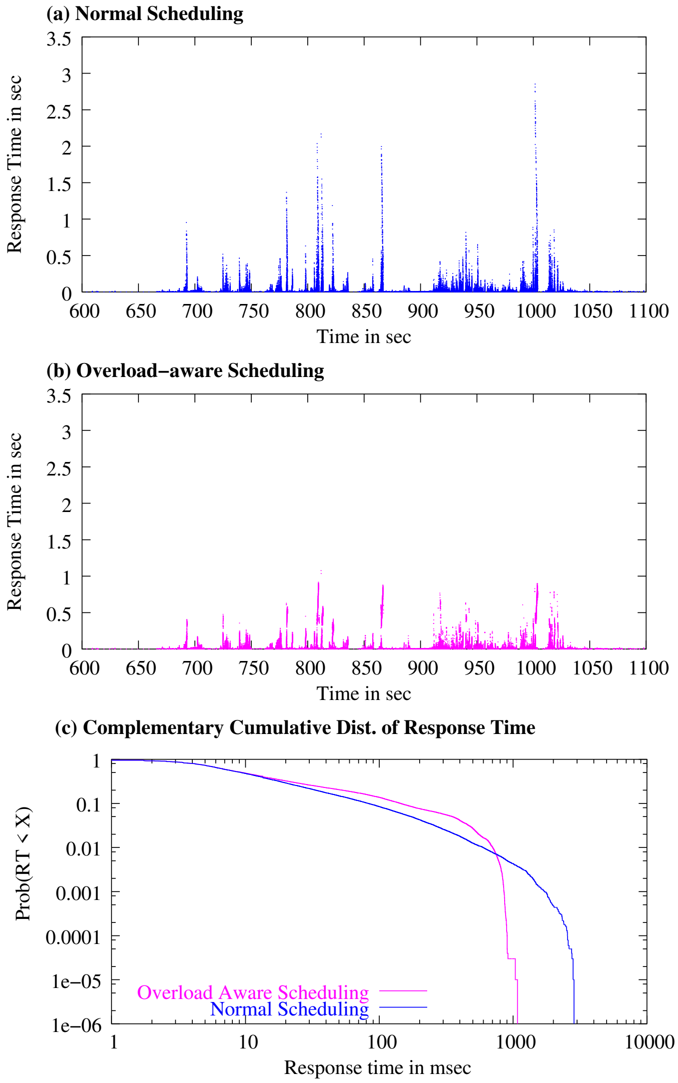


Fig. 6. Response time at the disk as a function of time under (a) normal scheduling, (b) overload-aware scheduling, and (c) the complementary cumulative distribution during the overwork period of experiment four.

source management at the various devices can significantly aid system performance. We have showed a first proof-of-concept that self-adaptive resource management at the lower tiers that can detect and handle overload and overwork cases can help in graceful performance degradation and in avoiding system unavailability. We are currently working on developing methods at the database level that can be be used to self-tune overload-sensitive memory allocation policies. Preliminary results are promising and indicate that simple statistical mechanisms that compress workload-related information can lead to the development of robust load detection and propagation prevention.

References

[1] T. Abdelzaher, K. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):80–96, Jan. 2002.

[2] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Specification and implementation of dynamic content benchmarks. In *5th IEEE Workshop on Workload Characterization(WWC-5)*, Nov. 2002.

[3] M. Arlitt, D. Drishnamurthy, and J. Rolia. Characterizing the scalability of a large web-based shopping system. *ACM Transactions on Internet Technology*, 1(1):44–69, 2001.

[4] J. Carlstrom and R. Rom. Application-aware admission control and scheduling in web servers. In *Proceedings of Infocom'02*, pages 824–831, New York, June 2002.

[5] L. Cherkasova and P. Phaal. Session based admission control: a mechanism for peak load management of commercial web sites. *IEEE Transactions on Computers*, 51(6):669–685, June 2002.

[6] S. Elnikety, E. Nahum, J. Tracey, and W. Zwaenepoel. A method for transparent admission control and request scheduling in e-commerce web sites. In *Proceedings of the 13th international conference on World Wide Web*, pages 276–286. ACM Press, 2004.

[7] G. R. Ganger, B. L. Worthington, and Y. N. Patt. The DiskSim simulation environment, Version 2.0, Reference manual. Technical report, Electrical and Computer Engineering Department, Cannegie Mellon University, 1999.

[8] D. Garcia and J. Garcia. Tpc-w e-commerce benchmark evaluation. *IEEE Computer*, pages 42–48, Feb. 2003.

[9] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal. Size-based scheduling to improve web performance. *ACM Transactions on Computer Systems*, 21(1):207–233, May 2003.

[10] A. Kamra, V. Misra, and E. Nahum. Controlling the performance of 3-tiered web sites: modeling, design and implementation. In *Proceedings of SIGMETRICS, the International Conference on Measurement and Modeling of Computer Systems*, pages 414–415. ACM Press, 2004.

[11] V. Kanodia and E. W. Knightly. Ensuring latency targets in multiclass web servers. *IEEE Transactions on Parallel and Distributed Systems*, 14(1):84–93, Jan. 2003.

[12] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of Ethernet traffic. *IEEE/ACM Transactions on Networking*, 2:1–15, 1994.

[13] D. McWherter, B. Schroeder, N. Ailamaki, and M. Harchol-Balter. Priority mechanisms for oltp and transactional web applications. In *20th International Conference on Data Engineering (ICDE 2004)*, Boston, MA, Apr. 2004.

[14] D. A. Menasce and V. A. F. Almeida. *Capacity planning for Web services: metrics models and methods*. Prentice Hall, Inc., NJ, 2001.

[15] A. W. Moore. *Measurement-based management of network resources*. PhD thesis - Number 528, Computer Laboratory, University of Cambridge, 2002.

[16] MySQL AB. *MySQL*. http://www.mysql.com.

[17] V. N. Padmanabhan and L. Qiu. The content and access dynamics of a busy web site: Findings and implications. In *Proceedings of ACM SIGCOM'02*, Sweeden, Aug. 2000.

[18] S. Pandey, K. Ramamritham, and S. Chakrabarti. Monitoring the dynamic web to respond to continuous queries. In *The Twelfth International World Wide Web Conference (WWW2003)*, Budapest, Hungary, May 2003.

[19] V. Paxson and S. Floyd. Wide-area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995.

[20] PHARM Project. *Java TPC-W Implementation Distribution*. http://www.ece.wisc.edu/ pharm/, Department of Electrical and Computer Engineering and Computer Sciences Department, University of Wisconsin-Madison.

[21] The Apache Software Foundation. *Apache Web Server*. http://www.apache.org.

[22] Transaction Processing and Performance Council. *TPC-W*. http://www.tpc.org.

[23] D. Villela, P. Pradhan, and D. Rubenstein. Provisioning servers in the application tier for e-commerce systems. 2004.

[24] VMWare INC. *VMWare Workstation*. http://www.vmware.com.

[25] T. Voigt and P. Gunningberg. Handling multiple bottlenecks in web servers using adaptive inbound controls. In *International Workshop on Protocols For High-Speed Networks*, Berlin, Apr. 2002.

[26] H. W.Cain, R. Rajwar, M. Marden, and M. H. Lipasti. An architectural evaluation of java tpc-w. In *The Seventh International Symposium on High-Performance Computer Architecture*, Jan. 2001.

[27] M. Welsh and D. Culler. Adaptive overload control for busy Internet servers. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems, (USITS'03)*, Seattle, WA, 2003.

[28] B. L. Worthington, G. R. Ganger, and Y. N. Patt. Scheduling for modern disk drives and non-random workloads. Technical Report CSE-TR-194-94, Computer Science and Engineering Division, University of Michigan, 1994.

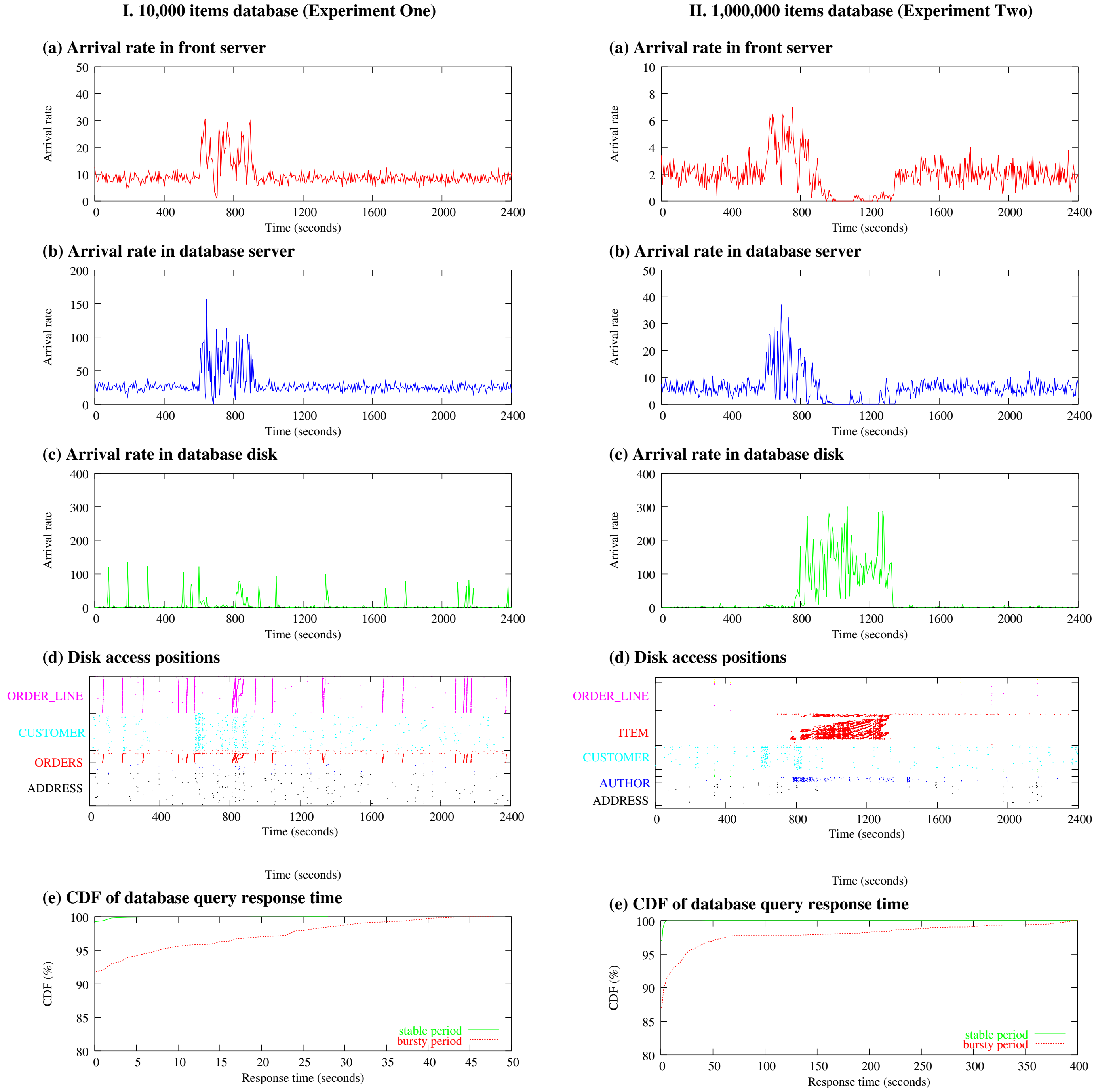


Fig. 2. Throughputs and utilizations in multiple tiers under transient load of the browsing mix.

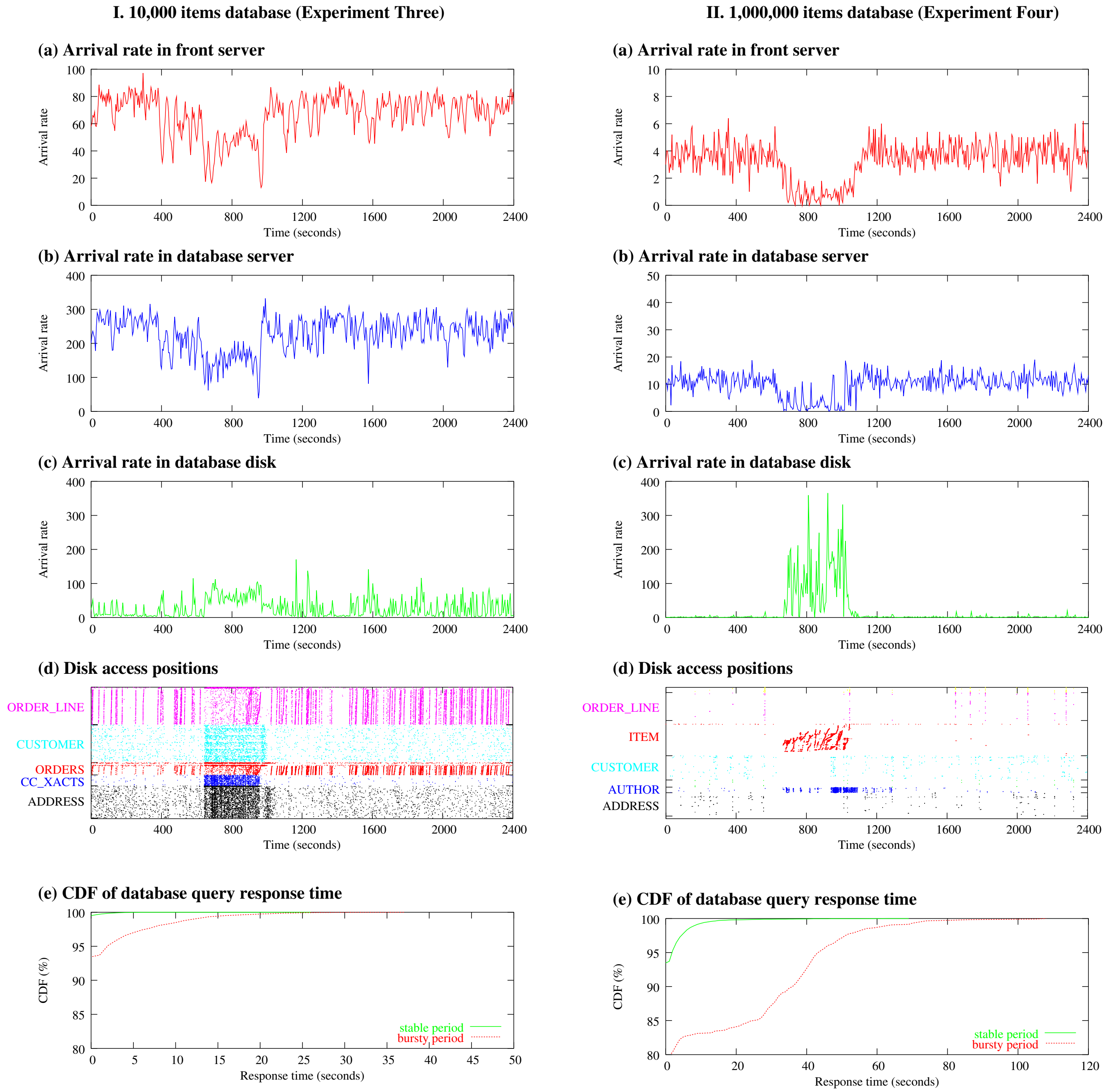


Fig. 3. Arrival rates and system utilizations in multiple levels under transient work, for the ordering mix and a variant (I) and the browsing mix and a variant (II).