On Choosing a Load-Balancing Algorithm for Parallel Systems with Temporal Constraints

Luís Fernando Orleans[1], Geraldo Zimbrão[1], Pedro Furtado[2]
[1]*COPPE, Department of Computer and Systems Engineering – Federal University of Rio of Janeiro, Brazil*
[2]*CISUC, Department of Informatics Engineering, University of Coimbra, Portugal*
{lforleans, zimbrao}@cos.ufrj.br, pnf@dei.uc.pt

Abstract

A key point in parallel systems design is the way clients requests are forwarded and distributed among the servers, trying to obtain the maximum throughput from them or, in other words, the load-balancing policy. Although it is a largely studied theme, with well accepted solutions, the inclusion of temporal constraints, also denoted as deadlines in this work, to the requests brings new complexities to the load-balancing problem: how to distribute the tasks and minimize the miss rate. The experiments describe along this paper attests that the workload variability plays a crucial role in this problem, pointing the big requests as the most critical elements. Our results also shows that even dynamic load-balancing algorithms are not able to reach an acceptable miss rate, since they handle both short tasks and big tasks the same way. Hence, we propose a new load-balancing algorithm, called ORBITA, which has a request identification and classification mechanism and an admission control module as well, restricting the number of big tasks within the system. This algorithm outperforms its competitors, which means that it has a bigger rate of tasks that end within the deadline, specially when the system is under high load. A prototype was also built in order to check the correctness of the simulation phase. The experiments were run against a benchmark tool, TPC-C, and all the results confirmed the previous assumptions, leading to the conclusion that it is a good practice to understand the system's workload in order to minimize the miss rate.

Keywords: load-balancing, parallel processing, deadline, ORBITA

1. Introduction

In parallel request processing systems, several parallel servers compute the incoming requests (or tasks) that are dispatched to them according to a load-balancing algorithm. Typically, these servers provide no guarantees about the response times for the request executions, in a so-called a *best-effort* approach. In peak situations, with requests arriving at high rates, this policy can lead to a scenario where a request takes tens of times longer to execute than it would take in a less stressed server. This way, if the system provides some kind of quality of service, such as trying to guarantee that response times would not be higher than an acceptable threshold, denoted as deadlines in this paper, the best-effort policy cannot be applied. Guaranteeing acceptable response times in parallel processing systems through load-balancing is the main objective of this paper. Our approach is meant to be applicable in different environments, including Transaction Processing Systems, Web Services, Virtualization Platforms.

This work proposes a new load-balance algorithm, based on the tasks durations (which are supposed to be known *a priori*), and our experiments prove that this is a better approach than blindly dispatching the tasks taking no further considerations – as most load-balance algorithms do. Although there already exists size-aware load-balancing algorithms, such as SITA-E [20], they do not comprise response times concerns, which is responsible for their poor performance on stressed systems with deadlined-tasks.

In order to find the best alternative, we have to analyze the impact of deadlined-tasks and their variability in the known load-balancing techniques. Our approach is valid and performs better than traditional load-balancing ones for both hard or soft deadlines.

The simulated architecture comprises only two servers, because it is the simplest possible parallel architecture. This can be easily expanded to *n* servers as well and this generalization will be discussed throughout the paper.

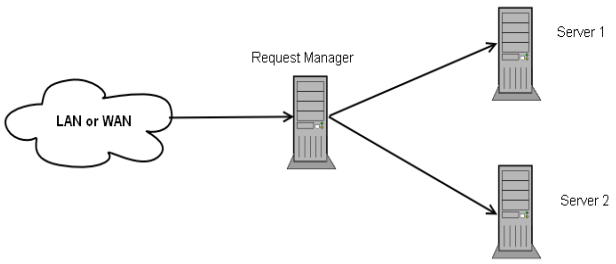

Figure 1: Simulated architecture

2. Related Work

There are plenty of works about load-balancing and QoS, most of them leading to already accepted and consolidated conclusions. Although these are almost exhausted themes, their combination seems to be an area where there are very few research results.

2.1. Load-balancing

Many load-balancing algorithms have been studied, most of them trying to maximize throughput or minimize the mean response time of requests. Reference [19] proposes an algorithm called TAGS, which is supposed to be the best choice when task sizes are unknown and they all follow a heavy-tailed distribution. This is not the case for the scenario analyzed in this paper, in which task sizes must be below a deadline threshold. It is also shown in [19] that, when task sizes are not heavy-tailed, Least Work Remaining has a higher throughput then TAGS. In fact, [24] and [23] claim that Least-Work Remaining is optimal when task sizes are exponential and unknown.

The algorithm SITA-E [20] has the best performance when task sizes are known and heavy-tailed but, otherwise, Least-Work-Remaining presents a better throughput.

Our previous work in [25] presented a technique to determine the best multiprogramming level (MPL) *offline*. Such concept had been expanded and we propose an algorithm that computes the maximum MPL in *runtime*.

2.2. Quality-of-Service (QoS)

In real distributed systems, task sizes are heavy-tailed. This means that a very small portion of all tasks are responsible for half of the load [21]. Most tasks are very small and there is a small number of big tasks as well. In models with deadlines, like the one analyzed in this paper, a similar distribution occurs.

Reference [30] presents a model where the number of concurrent requests within the system is restricted. When this number is reached, the subsequent requests are enqueued. But this model has no concern for deadlines or rejection of requests. It also does not show a way to load-balance the arriving tasks, since it is a single-server architecture.

Quality-of-Service was also studied for Web Servers. In [9] the authors propose session-based Admission Control (SBAC), noting that longer sessions may result in purchases and therefore should not be discriminated in overloaded conditions. They propose self-tunable admission control based on hybrid or predictive strategies. Reference [8] uses a rather complex analytical model to perform admission control. There are also approaches proposing some kind of service differentiation: [5] proposes architecture for Web servers with differentiated services; [6] defines two priority classes for requests, with corresponding queues and admission control over those queues. In [30], the authors propose an approach for Web Servers to adapt automatically to changing workload characteristics and [14] proposes a strategy that improves the service to requests using statistical characterization of those requests and services.

Comparing to our own work, the load-balancing alternatives referred above do not consider QoS parameters, such as deadlines, and the QoS studies concern single server systems, only. Our approach uses multiple servers and a careful request allocation to those servers in order to comply with the deadline constraints.

3. Modelling typical real distributions

In order to analyze and propose time-considering load-balance approaches, it is important to understand first the kinds of workloads distributions that happen typically and how to model them. The typical request workload, such as Transaction Processing Systems, is quite heterogeneous in what concerns servicing requirements.

Besides the algorithm SITA-E, reference [20] presents a study that claims that the distribution of task sizes (or durations) in computer applications are not exponential, but heavy-tailed. In short, a heavy-tailed distribution follows three properties:

1. Decreasing failure rate: the longer a task runs, the longer it is expected to continue running.
2. Infinite variance
3. A very small fraction (< 1%) of the very largest tasks makes up a large fraction (50%) of the load. This property is often called as the *heavy-tailed property*.

The simplest heavy-tailed distribution is the *Pareto* distribution, with probability mass function:

$$f(x) = ak^a x^{-a-1}, a, k > 0, x > k \quad ,$$

and cumulative distribution function

$$F(x) = 1 - (k \div x)^a \quad .$$

In these functions, k is the smallest possible observation, whereas a is the exponent of the power law, and will be called hereafter as the *variance factor* of the function. It varies from 0 to 2 and the more it is close to 0, the greater is the variability.

4. Traditional load-balancing algorithms and their weakness

Load balancing is a fundamental basic building block

for construction of scalable systems with multiple processing elements. There are several proposed load-balancing algorithms, but one of the most common in practice is also one of the simplest ones - Round-Robin (RR). This algorithm produces a *static* load-balancing functionality, as the tasks are distributed round-the-table with no further considerations. At the other hand, the algorithm Least-Work-Remaining (LWR) produces a *dynamic* load-balancing, as the arriving tasks are dispatched to the server with the least utilization (jobs on queue or concurrent executing tasks). This algorithm is considered in this study, as it is supposed to be the best choice when tasks durations are not heavy-tailed. According to [19] and [20], the main issue involving these algorithms is the absence of a control over big tasks, mixing inside the same server short (small) and long (big) tasks. It becomes more evident when tasks durations are heavy-tailed, with a minuscule fraction of the incoming tasks being responsible for half of the load. In fact, both [19] and [20] propose size-aware algorithms, trying to minimize the effects caused by the big tasks on the small ones.

We are concerned with guaranteeing specified acceptable response time limits. The number of concurrent executions (CE) is a crucial variable when deadlines are involved, because as we increase the number of CE we have a larger probability of missing the deadlines. As we are going to see, this is an issue that affects mostly systems where tasks durations have a high variability, which means that the occurrence of big tasks is more usual. When the variability is low, i.e., the number of big tasks is near to zero, all algorithms have similar performance curves and practically all tasks are completed. As performance starts to degrade as the variability begins to increase, the number of canceled tasks also gets higher, which gives space for a new size-aware load-balance algorithm, On-demand Restriction for Big Tasks, or ORBITA in short. The main idea is to separate the short tasks, which will always be submitted to execution, from the big tasks, which will have their admission by a server (or node) dynamically controlled. This way, a node will only admit big tasks that will not make the other already running big tasks miss their deadlines. Otherwise, the big task will be rejected by the node, as its admittance would lead to further performance degradation. In an *n* servers scenario, a task is only rejected if none of the *n* servers is able to handle it.

In the following we describe each load balancing algorithm we compare considering deadlines and rejection:

4.1. Least-Work-Remaining (LWR)

```
for each task that arrives:
  next_server := server_list ->least_utilized
  send (task, next_server)
```

4.2. Task Assignment by Guessing Size (TAGS)

In this algorithm, all incoming tasks are dispatched to the first server. If a task is running for too long, i.e., is a big task, it is killed and restarted from scratch on the second server.

```
for each task that arrives:
  send (task, first_server)
  schedule_dispatch_to_second_server(task)
```

4.3. Size Interval Task Assignment with Equal Load (SITA-E)

```
for each task that arrives:
  if task is a big task
    server := server_list ->second_server
  else
    server := server_list ->first_server
  send (task, server)
```

4.4. On-demand Restrictions for Big Tasks (ORBITA)

```
for each task that arrives:
  if task is a small task
    server := server_list ->first_server
    send(task, server)
  else
    server := server_list ->second_server
    bigger_task := bigger_running_task(server)
    max_ce := LOWER_BOUND(deadline/bigger_task)
    if number_of_running_tasks(server) >= max_ce
      NOT_ADMITT(task)
    else
      send (task, server)
```

Figure 2 depicts how LWR and SITA-E behave. In the figure, a new job with estimated duration of 3 units of time (UT) is received (2a) and there are 2servers: one is executing 3 short tasks and the other one is executing 2 long tasks. If the load-balancer module, the light-gray rectangle in figures, is using a LWR strategy, then the new task is dispatched to the second server (2b). On the other hand, if a size-aware algorithm (like SITA-E) is used, the job is forwarded to the first server, the one containing short tasks (2c).

5. Simulation setup

Due to the large number of parameters involved, it becomes necessary to formally describe the simulation model used in this work. The simulator has the following parameters :

- Number of servers.
- Tasks arrival rate (follows an exponential distribution).

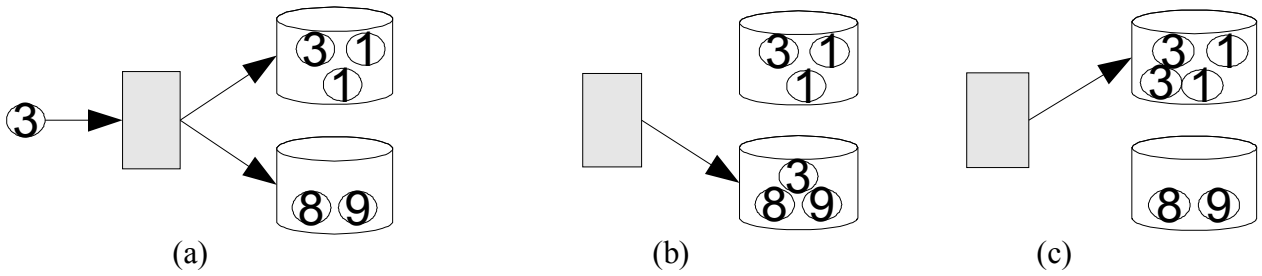

Figure 2: A new transaction arrives (a). LWR approach (b). Size-aware approach(c).

Table 1: Percentage of generated durations, according to the variance factor (*a*) of the Pareto distribution and the duration inteval.

a	[0, 1[	[1, 2[	[2, 3[	[3, 4[	[4, 5[	[5, 6[	[6, 7[	[7, 8[	[8, 9[	[9, 10[	10
0,1	53,88%	6,97%	5,17%	4,90%	4,42%	4,64%	3,93%	4,24%	4,25%	3,74%	3,85%
0,2	76,02%	4,75%	3,18%	2,94%	2,30%	1,90%	1,87%	1,95%	1,79%	1,56%	1,75%
0,3	87,90%	2,91%	1,76%	1,38%	1,12%	0,98%	0,89%	0,80%	0,78%	0,75%	0,73%

- Task size distribution (follows a Pareto distribution).
- Maximum amount of time a task can execute (deadline).
- Load-balancing algorithm.
- Minimum size of "big tasks".

In this paper, a system with 2 identical servers was simulated. The deadline time was set to 20 seconds and the duration of each request follows a Pareto distribution, where the value of the parameter *a* varies from 0.1 through 0.3, step 0.1, the smallest possible duration is 0.001 second (1 millisecond) and the highest duration is 10 seconds. In addition, all tasks have the same priority. The concurrency model is linear, which means that a task will take twice longer if it shares the server with another task, it will take three times longer in case of two other tasks and so forth.

The simulator implements all the described algorithms: TAGS, SITA-E, LWR and ORBITA and task arrivals follow an exponential distribution, with λ varying from 1 to 10, step 1. Finally, the tasks which have their durations below 1 second are considered small tasks. The big tasks are constituted by all the other durations. To eliminate the transient phase, the data obtained in the first hour of the simulation was discarded. Only the results obtained in the next 5 hours were considered.

5.1. Task duration generation

Since we are simulating a scenario where tasks have deadlines, the variance is not infinite. But as we are interested in studying how the system behaves when the number of small tasks is much greater than the number of big tasks, the Pareto distribution is used to generate the duration of the tasks. The MOD function will be applied to all durations that exceed the deadline time (generated_duration MOD deadline), in order to equally distribute them among the allowed durations. Table 1 shows the percentage of the generated durations for values of *a* versus the intervals (that must be read as [*min*, *max*[). It is to notice that when *a* assumes lower values, the variability is higher. Even in these cases, the number of durations within the interval [0,1[is much greater than the others.

5.2. Simulation Results

In these experimental results, we analyze first results for all tasks, showing that ORBITA has better or at least as good performance as other approaches in that case, and almost zero miss rates unlike the other strategies. The big tasks are a small fraction of the workload, but they are the ones with the largest miss rates for most strategies, and that is where ORBITA obtains much better results than the other ones because it considers time constraints. For this reason we then analyze results concerning the big tasks.

Figure 4a shows the performance of the algorithms when tasks durations highly vary. It can be noticed that, as the arrival rate raises, both LWR and TAGS performs worse in comparison with the other two algorithms, SITA-E and ORBITA. This low performance occurs due to the fact that those algorithms mix small and big tasks inside the same server, while SITA-E and ORBITA reserve a server to execute small, fast requests. A quick look to figure 5a attests this explanation: LWR and TAGS have small tasks getting canceled, an event that does not occur nor in SITA-E neither in ORBITA. Even a high arrival rate such as 10 tasks per second does not make the small tasks miss their deadlines when those algorithms are used.

If the variance factor was set to 0.3, the throughput of the four strategies would be very similar, as shown in figure 4b. As this variance factor generates a smaller number of big tasks, the assumption that mixing all kind

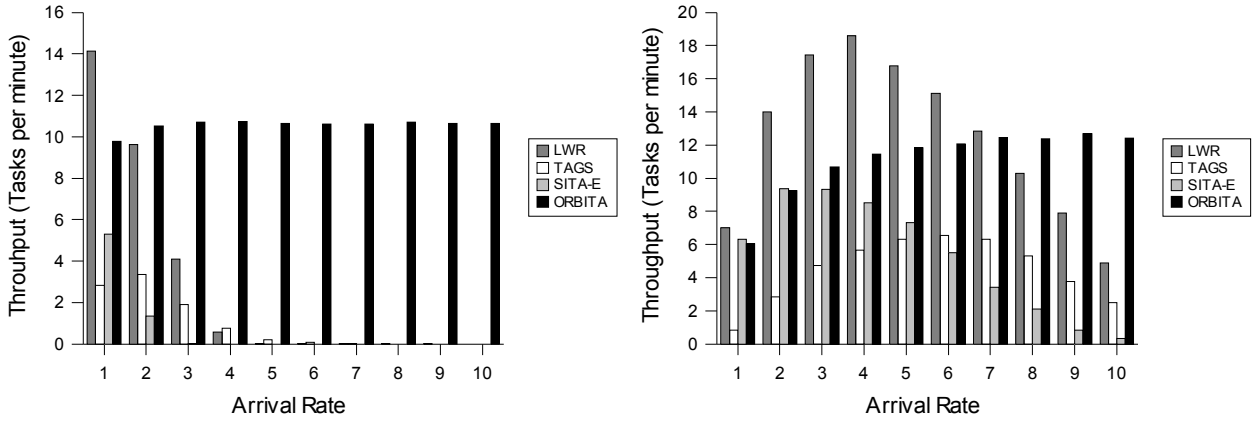

Figure 3: Throughput of big tasks with variance factor 0.1 (a) and 0.3 (b).

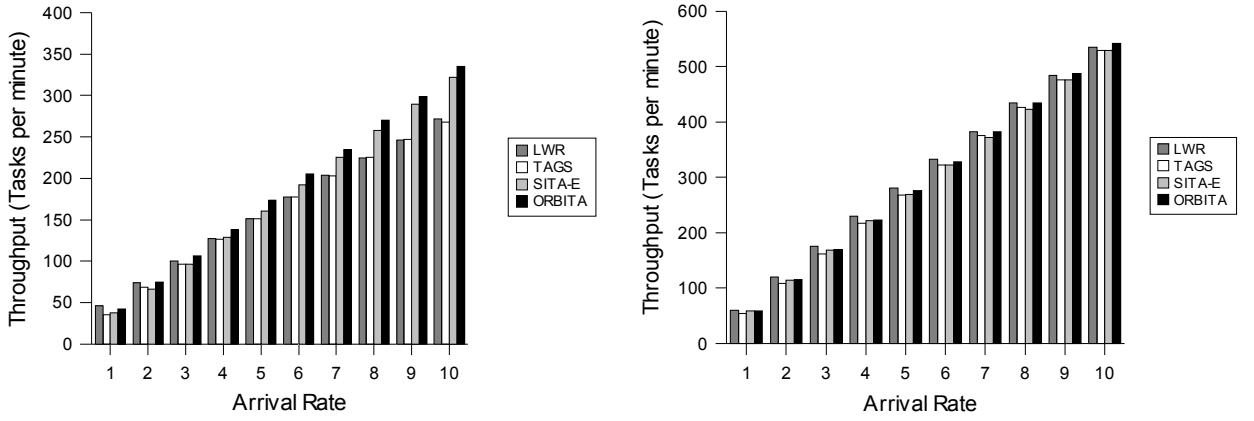

Figure 4: Total throughput with variance factor 0.1 (a) and 0.3 (b).

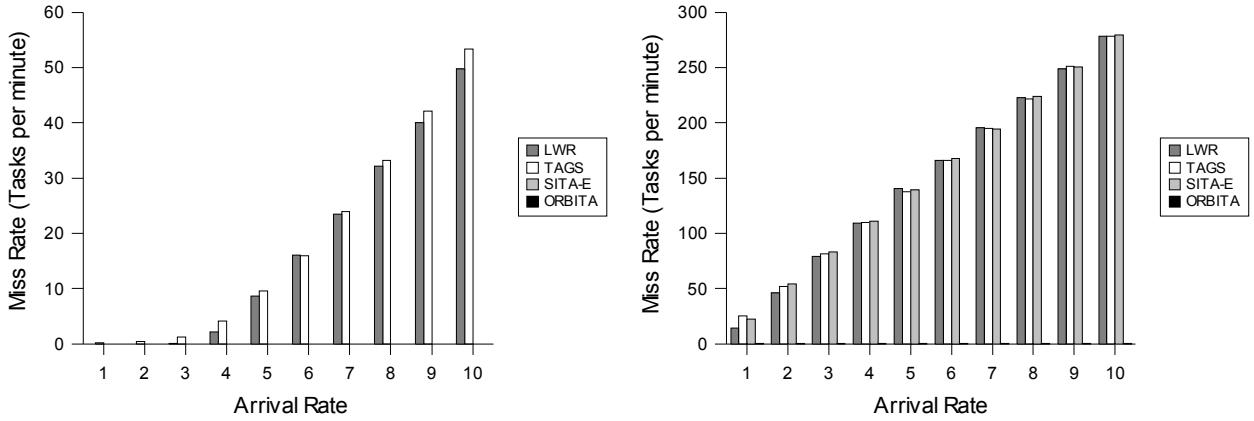

Figure 5: Throughput of small tasks with variance factor 0.1 (a) and 0.3 (b).

of tasks inside the same server for highly heterogeneous workloads is a bad idea, as shown in figure 4a, is reinforced.

It is worth noticing that in both cases in figure 4, the ORBITA algorithm has a better performance than its competitors. It becomes more evident if we analyze the throughput of small tasks and big tasks in separated graphics (figures 3 and 5).

As discussed above, both ORBITA and SITA-E policies prevent small tasks from being killed. This statement can be observed in figure 5a. A consequence of this event is that if we expand the simulated architecture from 2 to *n* parallel servers, only 1 server should be sufficient to handle all small tasks, with the other (*n-1*) servers being used to handle the big tasks. If the arrival rate keeps growing, there will be a moment when the number of servers destinated to handle small tasks should also increase. But the point is that the number of nodes that should be used to handle big tasks is much more critical than those that should be dedicated to small tasks.

The graphics shown in figures 3a and 3b shows the throughput of the big tasks. These pictures confirm the robustness of ORBITA, which maintains the throughput of big tasks almost unaltered, even when the variance factor is 0.1 and the arrival rate is 10 tasks per second.

Figure 3a shows that, for highly heterogeneous workloads, ORBITA is the only strategy with satisfactory results.

In figure 3b, the algorithm LWR presents a better throughput than ORBITA for arrival rates below 7 tasks per second. A closer look at the ended tasks of each strategy, displayed in tables 2 and 3, shows that LWR strategy has a better throughput for task arrival rates comprehended between 1 and 6, and ORBITA presents a better throughput for arrival rates higher than 7 tasks per second. Considering results of figures 3a and 3b together, we conclude that ORBITA is better with highly heterogeneous workloads or high arrival rates due to its tight control on time constraints. LWR is also an interesting algorithm as it tries to optimize resource usage, but does not have enough control over time constraints. Our current and future work on this issue, involves considering adaptable ORBITA/LWR alternatives and LWR with time constraints.

6. Prototype experiments

The Midas middleware [29] is a tool that intercepts the requests sent by an application to a database server. It uses the Proxy Design Pattern, thus providing a transparent admission control layer, without requiring deep source code modifications. A simplified class-diagram is shown in figure 6.

The utilization of more than one database server opens the necessity to keep data synchronized on all nodes, an issue that is known as data replication. According to [35], there are two replication models: eager replication, where the updated data is synchronized at the other nodes before transaction completion, and lazy replication, where the updated data is sent to other servers after transaction completion. For simplicity, we used a primary copy replication strategy, since reference [35] attests it is the best choice for eager replication.

7. Experiment Setup

To better understand the implications of ACID properties on load-balancing algorithms, we performed various rounds of experiments using the TPC-C

Table 2: Throughput histogram of LWR algorithm with variance factor 0.3.

Arrival Rate	[0, 1 [	[1, 2 [	[2, 3 [	[3, 4 [	[4, 5 [	[5, 6 [	[6, 7 [	[7, 8 [	[8, 9 [	[9, 10 [	10
1	52,85	1,77	1,09	0,78	0,65	0,5	0,57	0,45	0,41	0,43	0,36
2	106,56	3,62	2,16	1,66	1,39	1,2	0,98	0,97	0,82	0,63	0,58
3	157,52	5,16	3,08	2,54	1,91	1,34	1,15	0,78	0,69	0,46	0,33
4	210,77	7,31	4,4	2,86	1,58	0,96	0,57	0,36	0,25	0,16	0,13
5	264,27	9,19	4,51	1,74	0,7	0,27	0,17	0,09	0,05	0,02	0,02
6	317,21	10,58	3,47	0,79	0,19	0,06	0,02	0,02	0	0	0
7	369,7	11,01	1,63	0,16	0,02	0,01	0	0	0	0	0
8	423,7	9,71	0,56	0,03	0	0	0	0	0	0	0
9	476,46	7,76	0,16	0	0	0	0	0	0	0	0
10	529,75	4,88	0,02	0	0	0	0	0	0	0	0

Table 3: Throughput histogram of ORBITA algorithm with variance factor 0.3.

Arrival Rate	[0, 1 [	[1, 2 [	[2, 3 [	[3, 4 [	[4, 5 [	[5, 6 [	[6, 7 [	[7, 8 [	[8, 9 [	[9, 10 [	10
1	53,06	1,43	0,91	0,69	0,53	0,5	0,44	0,41	0,41	0,37	0,37
2	106,07	2,31	1,29	1,11	0,87	0,81	0,64	0,65	0,57	0,55	0,45
3	159,01	2,6	1,38	1,23	1,03	0,97	0,82	0,71	0,68	0,68	0,59
4	211,17	2,92	1,73	1,2	1,05	0,95	0,85	0,8	0,68	0,64	0,65
5	263,86	2,9	1,85	1,24	1,02	1,08	0,86	0,72	0,79	0,65	0,73
6	316,49	3,02	1,75	1,3	1,12	0,94	0,94	0,9	0,71	0,67	0,72
7	370,23	3,21	1,98	1,35	1,11	0,97	0,88	0,75	0,76	0,73	0,71
8	422,01	3,08	1,75	1,47	1,07	1,05	0,9	0,87	0,81	0,7	0,68
9	474,35	3,27	1,98	1,28	1,24	1,08	0,9	0,83	0,74	0,7	0,68
10	529,24	3,13	1,8	1,29	1,11	1,02	0,93	0,91	0,82	0,72	0,68

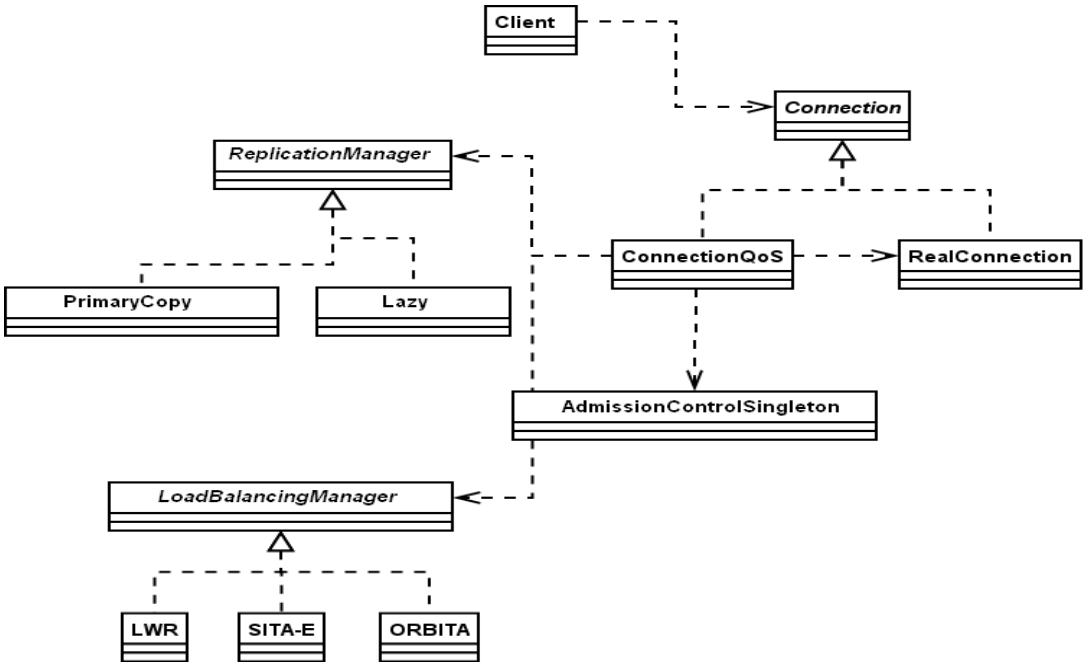

Figure 6: Simplified class-diagram of the Midas middleware.

benchmark (http://www.tpc.org/tpcc/) in a 2-servers full-replicated database.

Also, to create a more realistic scenario, we modified the workload generation. According to related work [19], typical transactional workloads present high-tailed properties, which is not the case of TPC-C's default workload. We also made an extra modification on the TPC-C's code, simulating an open model, i.e., transactions are sent to the system according to a statistical distribution (e.g. an Exponential Distribution). For more informations about open and closed simulation models, please refer to [32].

The use of an open model was a bit imprecise due to the high cost to create and destroy threads (for emulating clients). For arrival rates greater than 14 transactions per second, the mean value of the exponential distribution was not able to be reached. Nevertheless, the obtained results clearly simulated high-utilization scenarios, the aim of the experiments.

7.1. Workload Generation

As mentioned before, we used 2 types of transaction mixes: the one described in the TPC-C specification (which we call default transaction mix in this paper) and another one that aims to reflect a more realistic one, as stated in [19].(denoted here as heavy-tailed transaction mix).

Briefly explaining, the TPC-C specification proposes 5 different transactions (in parenthesis are their frequency of occurrence): new order (45%), payment (43%), delivery (4%), order status (4%) and stock level (4%). When executing in standalone mode, we found that new order transaction was the longest one and, in contrast, the stock level was the fastest transaction. So, we create the heavy-tailed transaction mix comprising only stock level (95%) and new order (5%) transactions, thus attending the requirements of a heavy-tailed distribution.

To classify transactions in short or long we used a map. As TPC-C has only five different transactions, we could store pounded mean response times (PMRT) for each one of them. Hence, each map entry was a pair {transaction name, PMRT}. If the PMRT value was greater (lower) than a threshold (1 second, arbitrarily chosen) then its corresponding transaction would be classified as long (short).

A last remark on the workloads: order status and stock level transactions are *read-only*, which means that they should not acquire any locks. On the other hand, the others are update transactions and their isolation levels were set to *Read-Committed.*

7.2. Experiments Details

All experiments were executed using a Pentium 4 3.2GHz, with 2GB RAM DDR2 and a 200 GB SATA HD which was responsible for creating the threads that simulate the clients. The servers were 2 Pentium II MMX 350MHz, with 256MB RAM and a 60GB IDE HD. Both servers were running a Debian Linux, with Kernel version 2.6 and were connected by a full-duplex 100Mbps Ethernet link. A PostgreSQL 8.1 database server was running on each server machine and the database size was 1.11GB. The client machine used a

Sun Microsystems' Java Virtual Machine, version 1.5. The database was created with 10 warehouses.

The reason for using the slower computers as the database servers relies on our need to stress the system. Our intention is not to maximize the throughput within deadline (TWD) – the most important metric for this work, but to analyze the impact of ACID properties on load-balancing algorithms and their implications on QoS constraints.

Table 4: Simulation parameters and their values

Parameter	Value
Arrival rate	Exponential distribution
Simulation time	20 minutes
Warm-up time	5 minutes
Deadline	5 seconds

Finally, each round of experiments was executed for a period of 20 minutes. During the first 5 minutes no data was collected. Before each round, the database was dropped and then recreated, guaranteeing that all rounds of experiments used the same database state.

7.3. Results

Figures 7 and 8 show the throughput of transactions that ended within the deadline versus the arrival rate. For best visualization, the results are displayed in terms of long transactions (a) and short transactions (b).

It can be seen that when the default workload is used (figures 7a and 7b), LWR performs badly. As a result for mixing inside the same server both short and long transactions, only those which are the fastest ones are able to end within the deadline. This can be explained by the high number of locks obtained by update transactions (the majority on this workload), interfering on the execution of read-only transactions.

On the other hand, SITA-E and ORBITA performs better due to their size-aware nature. Such algorithms estimate the duration of incoming transactions and, if classified as long (short), the transaction is forwarded to the appropriate server. The thing to be noticed here is that short transactions are the read-only ones, thus they do no acquire locks. This explains the high number of short transactions that were executed within the deadline with these algorithms. On the other hand, the TWD of update transactions is smaller, and for SITA-E it even presents a decreasing form on figure 7a. Why does ORBITA perform better than SITA-E? The reason is the admission control that ORBITA provides for long (big) transactions – which are responsible for acquiring locks and, in these cases, for a long period of time. Controlling the admission of long transactions directly implies in controlling the number of locks, thus keeping the accepted transactions ending their executions within their deadlines. It can be seen from figure 7a that an arrival rate of 10 transactions per second reaches the optimal TWD.

A last analysis of figure 7 that is worth mentioning is about replication. Once all update transactions are replicated to others servers, why the performance of small transactions was not affected by the replication in SITA-E and ORBITA? The most reasonable answer for this question relies on the assumption that only the write-set (WS) of update transactions were forwarded to the other server. These WS's do not contain any 'select' statements, thus their execution is also very fast – which means that their executions do slow down the read-only transaction, but are not capable for making them to miss their deadlines.

Figures 8a and 8b present the performance of the same algorithms when a heavy-tailed workload is used. One of the properties of such a distribution is massive presence of short transactions and only a few of very, very big transactions. The results in these figures are interesting, because the presence of only 5% of new order transactions is sufficient to reduce a lot the TWD of short transactions when the LWR algorithm was chosen. In fact, when the arrival rate is 6 (which means 360 transactions per minute, 95% of them are short), TWD reached its maximum value, about 100. As both servers have to execute new order transactions, locks used by this transaction (which includes a 'select for update' clause) also occur everywhere. Again, the TWD of long transactions is zero – even for small arrival rates. This can be ironically explained by the massive presence of fast transactions, which severely interferes on the executions of long transactions. So, long transactions do not permit that the short ones execute within deadline. In turn, the presence of lots of short transactions are responsible for the zero-TWD of long jobs.

The other algorithms, SITA-E and ORBITA, have better performances. As in the default workload setup, none of the small transactions missed their deadlines, even when data modifications provided by the replications occur. On the other hand, the execution of long transactions for SITA-E becomes critical. As the arrival rate increases, the TWD fastly decreases until it reaches zero. Again, the high number of locks are responsible for the poor performance. In contrast to SITA-E, ORBITA is capable to maintain the TWD of long transactions in its maximum, due to its admission control mechanism. A remark about this admission control relies on the reduced number of long transactions admitted. When TWD reached its maximum value, the arrival rate is about 10, what means that 600 transactions arrives per minute – 5% of them (about 30) are of new order type – but only 7 of them were admitted. Otherwise, there might be conflicts on lock acquisitions and, probably, some transactions (maybe even all of them) would miss their deadlines.

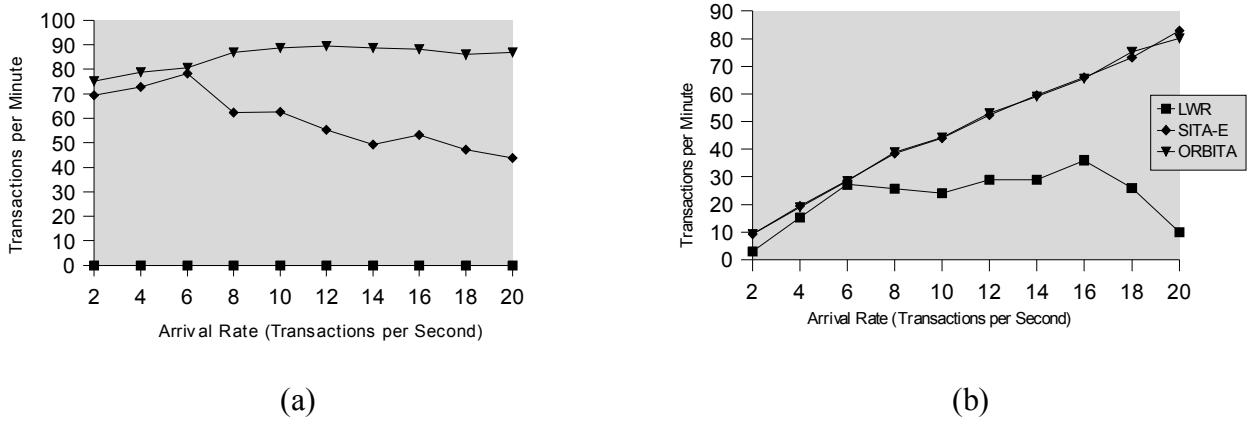

Figure 7: Throughputs of long transactions (a) and small transactions (b) when default workload is used.

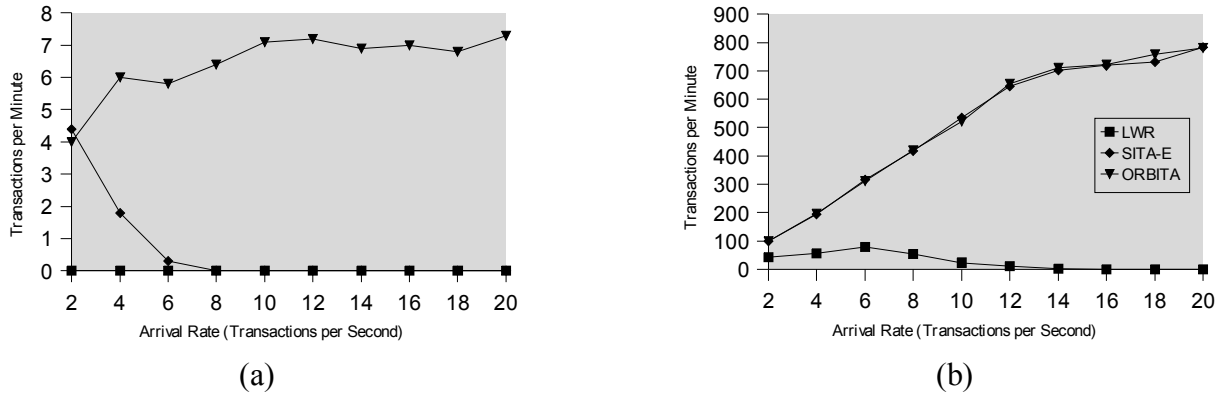

Figure. 8: Throughputs of long transactions (a) and small transactions (b) when heavy-tailed workload is used.

8. Conclusion and future work

In a parallel system, the incoming tasks must be dispatched to a server according to a load-balancing algorithm. Most of these algorithms are not aware about the response times of the tasks, since they all follow a best-effort policy. Thus, if tasks arrive at a very high rate, not only fast requests will have to wait for a long period to be executed, as slower requests will take too long. In this paper we propose ORBITA, a load-balance algorithm that takes time into consideration.

The ORBITA algorithm, which is based on the assumption that tasks durations follow a highly heterogeneous distribution, differentiates service between small and large requests in order to provide time guarantees. It was experimentally proved that the big tasks were the ones responsible for deadline misses, so ORBITA works by separating the fast, small tasks from the big tasks, which have their admission controlled by each server. A big task will only be admitted into a server if it does not make the other running big tasks miss their deadlines.

The experiments have shown that when the variability of the task durations is high, ORBITA's throughput is not only greater than the other algorithms, but fairer, since tasks of all size intervals have low miss rates.

We also implemented a prototype containing three different of the presented load-balancing techniques, each one with different characteristics: Least-Work-Remaining (LWR), Size Interval for Task Assignment with Equal Load (SITA-E) and On-Demand Restrictions for Big Tasks (ORBITA), our proposal.

Those rounds of experiments were executed in a 2-servers fully replicated database. The dynamic algorithm (LWR) presented the worst performance, since it does not make differentiations on transactions. Thus, by mixing inside the same server update and read-only transactions, the isolation needed for the first group was responsible for making lots of transactions (of both groups) to miss their deadlines.

On the other hand, the size-aware algorithms (SITA-E and ORBITA) dynamically recognized read-only (update) transactions and classified them in short (long). This way, each group was assigned to a dedicated server, hence reducing the overall miss rate. Even the replication cost was not sufficient for deteriorate the performance of read-only transactions, since only the write-set of the whole update transaction was forwarded to the other server. The main problem for SITA-E relies on the execution of long transactions, which are responsible for acquiring the locks. The uncontrolled admission of update transactions present in the SITA-E can reduce the number of transactions ended within the deadline per minute to zero. In contrast, ORBITA has an admission control of big transactions. An update transaction is only admitted by the system if it will not cause deadline misses – from itself or from the other already-running transactions.

As future works, we intend to investigate how to effectively identify and estimate the duration of transactions. The solution adopted in this paper (using a map with pounded mean response times) was sufficient for what this work was intended, but we are concerned on if and how to generalize such a concept for a system with ad-hoc transactions. We also intend to work on database internals level and study the viability of adding time-constraints mechanisms to queries and/or transactions.

As future work we intend to investigate how ORBITA can be made to adapt automatically to actual workloads and arrival rates. This includes how to determine the number of servers needed to handle each type of tasks (big and small) and, instead of dividing tasks into two classes statically, how to determine and manage task classes automatically. We also intend to work on a time-constrained version of LWR and compare the approaches.

References

[1] Akal, F. et al (2005), Fine-Grained Replication and Scheduling with Freshness and Correctness Guarantees. 31st Very Large Databases Conference, p. 565 – 576, Trondheim, Norway, 2005.

[2] Amza, C., Cox, A.L., Zwaenepoel, W. (2005), A Comparative Evalution of Transparent Scaling Techniques for Dynamic Content Servers. 21st International Conference On Data Engineering, 2005.

[3] Barker, K. et al (2004), A Load-Balancing Framework for Adaptative and Asynchronous Applications, IEEE Journal Transactions on Parallel And Distributed Systems, v.15, n.2.

[4] Barker, K.; Chernikov, A.; Crhisochoides, N.; Pingali, K. (2004), A Load Balancing Framework for Adaptive and Asynchronous Applications. IEEE Transactions on Parallel and Distributed Systems, v. 15, n. 2

[5] Bhatti, N.; Friedrich, R. (1999) Web server support for tiered services. IEEE Network, v.13, n.5, p. 64–71.

[6] Bhoj; Rmanathan; Singhal. (2000) Web2K: Bringing QoS to Web servers. Tech. Rep. HPL-2000-61, HP Labs.

[7] Cardellini, V.; Colajanni, M.; Yu, P.S. (2002), The State of the Art in Locally Distributed Web-Server Systems. ACM Computing Surveys, v. 34, p.263 – 311

[8] Chen, X; Mohaptra, P; Chen, H.(2001), An admission control scheme for predictable server response time forWeb accesses. In Proceedings of the 10th World Wide Web Conference, Hong Kong.

[9] Cherkasova; Phall (2002), Session-based admission control: A mechanism for peak load management of commercial Web sites. IEEE Req. on Computers, v.51, n.6.

[10] Crovella, M.; Bestravos, A. (1997), A.Self-similarity in World Wide Web traffic: Evidence and possible causes. IEEE/ACM Transactions on Networking, p.835-836.

[11] Daudjee, K.; Salem, K. (2006), Lazy Database Replication with Snapshot Isolation, 30th Very Large Databases Conference, p. 715 – 726, Seoul, Korea.

[12] Devine, D.K. et al (2005), New challenges in dynamic load balancing,ADAPT '03: Conference on Adaptive Methods for Partial Differential Equations and Large-Scale Computation, v. 52, n. 2-3, p. 133 – 152

[13] Dyachuk, D.; Deters, R. (2007), Optimizing Performance of Web Service Providers, IEEE 21st International Conference on Advanced Information Networking and Applications, p. 46 – 53, Niagara Falls, Ontario, Canada.

[14] Elnikety, S.; Nahum, E.; Tracey, J; Zwaenepoel, W. (2004) A Method for Transparent Admission Control and Request Scheduling in E-Commerce Web Sites, WWW2004: The Thirteenth International World Wide Web Conference, New York City, NY, USA.

[15] Furtado, P.; Santos, C. (2007), Extensible Contract Broker for Performance Differentiation, International Workshop on Software Engineering for Adaptive and Self-Managing Systems, Minneapolis, USA.

[16] Gamma, E. et al (1994), Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley.

[17] Ghazalie, T. M.; Baker (1995), T. P. Aperiodic Servers In A Deadline Scheduling Environment. Real Time Systems Journal. v 9, n. 1, p. 31 – 67

[18] Gokhale, S. S.; Lu, J. (2006), Performance and Availability Analysis of E-Commerce Site, 30th Annual International Computer Software and Applications Conference, Chicago.

[19] Harchol-Balter, M. (2002), Task assignment with unknown duration.. Journal of the ACM

[20] Harchol-Balter, M.; Crovella, M.: Murta, C. (1999), On choosing a task assignment policy for a distributed server system. Journal of Parallel and Distributed Computing, v.59 n.2, 204-228.

[21] Harchol-Balter, M.; Downey, A. (1997), Exploiting process lifetime distributions for dynamic load-balancing. ACM Transactions on Computer Systems.

[22] Knightly, E.; Shroff, N. (1999), Admission Control for Statistical QoS: Theory and Practice. IEEE Network, v. 13, n. 2, pp. 20-29.

[23] Nelson, R.; Philips, T. (1993), An approximation for the mean response time for shortest queue routing with general interarrival and service times. Performance Evaluation, p.123-139.

[24] Nelson, R.; Philips, T. (1989), An approximation to the response time for shortest queue routing. Performance Evaluation Review, p.181-189.

[25] Orleans, L.F., Furtado, P.N. (2007), Optimization for QoS on Web-Service-Based Systems with Tasks Deadlines, ICAS'07 Third International Conference on Autonomic and Autonomous Systems, 2007.

[26] Orleans, L.F., Furtado (2007), P.N., Fair Load-Balancing on Parallel Systems for QoS, ICPP International Conference on Parallel Processing, p. 22

[27] Orleans, L.F., Zimbrão, G., Furtado, P.N. (2008), Controlling the Behaviour of Database Servers with 2PAC and DiffServ, DEXA'08 - 19th International Conference, DEXA 2008, Turin, Italy

[28] Orleans, L.F., Zimbrão, G., Oliveira, C.E.T. (2008), On Choosing a Load-Balancing Algorithm for Parallel Databases with Time-Constraints, SBBD'08, XXIII Brazilian Symposium on Databases, São Paulo, Brazil.

[29]. Orleans, L.F. (2007), "ORBITA: Uma Estratégia de Balanceamento de Carga para Tarefas com Restrições Temporais", M.Sc. Dissertation, NCE/UFRJ.

[30] Pradhan, P.; Tewary, R; Sahu, S; Chandra, A. (2002), An observation-based approach towards self managing Web servers. In International Workshop on Quality of Service, Miami Beach, FL.

[31] Schroeder, B.; Harchol-Balter, M. (2006), Achieving class-based QoS for transactional workloads. IEEE International Conference on Data Engineering.

[32] Schroeder, B.; Wierman, A.; Harchol-Balter, M. (2006), Open Versus Closed: A Cautionary Tale. Network System Design and Implementation, San Jose, CA. Pp 239-252.

[33] Serra, A.; Gaïti, D.; Barroso, G.; Boudy, J. (2005), Assuring QoS Differentiation and Load-Balancing on Web Servers Clusters. IEEE Conference on Control Applications, p. 8.85-890.

[34] Wiesmann, M. et al (2000), Understanding Replication in Databases and Distributed Systems, 20th IEEE International Conference on Distributed Computing Systems , p. 464.

[35] Wiesmann, M., Schiper, A. (2005), Comparison of Database Replication Techniques Based on Total Order Broadcast. IEEE Journal Transactions On Knowledge And Data Engineering, v. 17, n. 4, p. 551 – 566.

[36] Wydrowski, B.;Zukerman, M. (2002), QoS in Best-Effort Networks, IEEE Communications Magazine.

[37]. Xiong, M. et al (2002), Scheduling Transactions with Temporal Constraints: Exploiting Data Semantics, IEEE Journal Transactions On Knowledge And Data Engineering, v.14, n.5, p.1155 – 1166