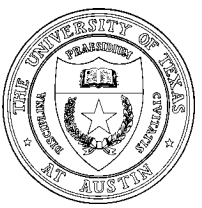
VIRTUALLY OWNED COMPUTERS – A NEW PARADIGM FOR DISTRIBUTED OPERATING SYSTEMS

Banu Özden
Avi Silberschatz
Aaron J. Goldberg

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-93-25 November 1993



DEPARTMENT OF COMPUTER SCIENCES
THE UNIVERSITY OF TEXAS AT AUSTIN

AUSTIN, TEXAS 78712

Virtually Owned Computers—A New Paradigm for Distributed Operating Systems *

Banu Özden

Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, Texas 78712-1084

Avi Silberschatz †

Aaron J. Goldberg

600 Mountain Ave.
AT&T Bell Laboratories
Murray Hill, NJ 07974

Abstract

Existing distributed operating systems lack two key features— *predictability* and *choice.* Predictability refers to the ability of the system to provide each user with a computing environment whose performance is independent of the behavior of other users. Choice refers to the ability of a user to select a computer system that meets that user's specifications, needs or budget. In this paper, we introduce the *virtually owned computers* (VOC) paradigm that allows one to incorporate these concepts into the design of distributed operating systems. In a distributed system based on the VOC paradigm, each user is promised a given quality of service, and the system seeks to provide each user with at least that level of service. One can view the service promised to a user as a *virtual computer* owned by that user. Ultimately, a user should receive the promised service independent of the location where the actual execution takes place and where the user accesses the system. Different users may be promised different levels of service corresponding to different "types" of virtual computer. In order to support the VOC paradigm, many issues in the design of a distributed operating system must be reconsidered including resource management, naming, protection, and service provision. In this paper, we focus on scheduling issues. We demonstrate that existing scheduling algorithms for distributed and real time systems are not directly applicable to VOC systems and study the basic problems associated with scheduling under the VOC paradigm.

Index Terms

Distributed operating systems, load distribution, load sharing, owner-based systems, scheduling, real-time systems.

*The research of Banu Özden and Avi Silberschatz was supported in part by the Texas Advanced Technology Program under Grant No. ATP-024, the National Science Foundation under Grant Nos. IRI-9003341 and IRI-9106450, and grants from the IBM and Hewlett-Packard corporations.

†On leave from the Department of Computer Sciences, at the University of Texas at Austin.

1 Introduction

Personal computers provide two attractive features that are neglected in today's distributed operating systems—*predictability* and *choice.* Predictability is the ability of the system to provide each user with a computing environment whose performance is independent of the behavior of other users. Choice refers to the ability of a user to select a computer system that meets that user's budget, needs or desire. We refer to the pair of features as *ownership,* since these two features together represent the rights of the computer owner. Our goal is to add the concept of *ownership* to distributed systems.

A distributed system can be built on various system models which include the *processor-pool model*, the *workstation model*, and the *time-shared computers model* [TR85]. Though it was originally assumed that the processor-pool model would dominate distributed systems, current computing environments typically contain autonomous workstations connected through networks. A primary reason for the shift to such computing environments is that workstations provide users with ownership rights. A user can decide what workstation to purchase (choice) and whether or not to share the workstation with other users (predictability).

In order to incorporate the concept of ownership into the distributed operating systems, we propose a new paradigm—*virtually owned computers* (VOC). In a distributed system based on the VOC paradigm, each user is promised a given quality of service, and the system seeks to provide each user at least the level of service promised. One can view the service promised to a user as a *virtual computer* owned by that user. Ultimately, a user should receive the promised service independent of the location where the actual execution takes place and where the user accesses the system. Each user may have a different level of service promised; namely, a different type of virtual computer. The VOC paradigm can be realized on various system models including the processor-pool model, the workstation model, and the time-shared computers model. By providing the users of a distributed system with the desirable properties of personal computers—predictability and choice—the VOC paradigm encourages the users to share their computers with others, or to use a distributed system instead of buying their own private computers.

A distributed operating system based on the VOC paradigm will be referred to in this paper as a VOC system. In order to implement a VOC system, some of the issues in the design of a distributed operating system must be reconsidered including resource management, naming, protection, and service provisions. This paper concentrates on the processor scheduling aspects

of resource management.

The remainder of the paper is organized as follows. Section 2 places VOC in the context of previous distributed systems paradigms. Section 3 formally defines the VOC paradigm. In Sections 4-6, we focus on the scheduling problem in VOC systems, and compare the problems to the ones in traditional distributed and real time systems. In Section 7 we briefly describe the salient features of the Eagle distributed operating system which is currently being implemented at the University Of Texas at Austin. The Eagle system is based on the VOC paradigm. Finally, in Section 8, we offer some concluding remarks and note some open problems.

2 Related Work

In early distributed systems, the aim of load distribution is to share available computational power equally among all users. Some of these systems are based on the processor-pool model [NH82, Tan86, RPT90], whereas others are based on the workstation model [BL85, Stu88]. The concept of ownership in these systems either does not exist, or is not preserved. Thus, users of these systems cannot begin to predict how long their tasks will take to execute.

Recently, owner-based variants of the workstation model have been proposed where the aim of the load distributing algorithm is to share only the idle computational cycles that are not utilized by the owners of workstations [LLM88, KC91, DO91]. Ultimately, in these systems, workstation owners should be able to specify the policies that govern the conditions under which their workstations can be shared, and the load sharing algorithm will adhere to these policies [BOS93]. However, current owner-based systems do not provide predictable performance for an owner's tasks that are executing remotely on other machines. For example, when an owner starts using his workstation, foreign tasks (i.e., remote tasks of other owners) executing on that workstation are either migrated from the workstation as in Sprite and Condor [LLM88, KC91], or are continued but with reduced priority as in Stealth [DO91]. From the remote task owner's perspective, there are no guarantees of predictable performance for remote tasks.

The VOC paradigm can be viewed as an extension and a generalization of owner-based systems. It extends the owner-based systems in the sense that each user is promised a level of service. Further, the VOC paradigm generalizes owner-based systems in the sense that it can be implemented on system models other than the workstation model.

3 The VOC Paradigm

In the VOC paradigm, each user owns an imaginary computer—a virtual computer. The virtual computer is only a description of a computer and may not correspond to any real computer in the system. The description includes the CPU type (which includes CPU speed) and a local scheduling algorithm (e.g., first come first served (FCSF), round robin (RR)). The real computers used in the distributed system may be workstations, multiprocessors or mainframes depending on the system model. For our purpose, the system is simply modeled as a set $\mathcal{P} = \{P_1, P_2, ..., P_m\}$ of processors.

Let $\mathcal{V} = \{V_1, V_2, ..., V_n\}$ be the set of virtual computers. We assume there is one-to-one correspondence between the set of users and $\mathcal{V}$. The virtual computers of two different users may differ in terms of their specification. Assignment of different virtual computers to users will usually depend on factors such as a user's needs, seniority, and budget. For example, in an owner-based network of workstations, a user will be assigned a virtual computer that is equivalent to the user's own workstation.

Suppose that all of a user's tasks were executed on a computer which is equivalent to the user's virtual computer. Such an execution is referred to an *execution on the virtual computer.* We call a period in which no task is executing on the virtual computer an *idle period.* We call the period between two idle periods of a virtual computer a *burst* period. We say that a user *reclaims* his virtual computer at the beginning of each burst. The notion of reclamation can be defined in a number of different ways to support varying levels of sharing and predictability. We shall say more about this issue in Section 7.

The tasks that a user submits to the system are assumed to be submitted exclusively to the virtual computer of the user. The abstract execution of a task T_i on a virtual computer can be characterized by two attributes: arrival time a_i, and completion time $\bar{c}_i$ on the virtual computer. The arrival time is the time when T_i is submitted to the system. The completion time is the time when T_i would be completed if it were executed on a real computer equivalent to the virtual computer. Clearly $\bar{c}_i$ depends on the local scheduling algorithm of the virtual computer, the length of the task T_i, and the load of the virtual computer. We denote the actual completion time of T_i in the system by c_i.

A VOC system seeks to provide each user with a level of service which is at least as good as the one that could have been obtained on the user's virtual computer after the user has reclaimed it. That is, a VOC system ensures, whenever possible, that once a user reclaims his virtual computer,

for each task T_i belonging to that user, the condition $c_i \leq \bar{c}_i$ holds.

The abstraction of virtual computers generates a set of tasks $\mathcal{T} = \{T_1, T_2, ..., T_k\}$ that must be actually scheduled on the system. Three attributes are associated with each task T_i: arrival time a_i, deadline (time constraint) d_i, and processor constraint r_i. Deadline d_i is the time by which T_i must complete, namely $d_i = \bar{c}_i$. Processor constraint r_i is either a positive integer corresponding to the processor id on which the task must be executed (i.e., j if the task is to execute on processor P_j), or a negative integer to specify that the task can be executed on any processor. For example, if the task is a high bandwidth interactive application (e.g., a 3D CAD tool), then it may be constrained to execute on the processor that is connected to the graphical display.

In the VOC paradigm, the deadline of a task depends on the length of the task, CPU speed, the local scheduling algorithm, and the load of the virtual computer (the characteristics of tasks on the virtual computer). Since the load changes dynamically, the deadline of a task is not fixed at the arrival time, even if the length of the task is known.

A VOC system attempts to schedule all the tasks in $\mathcal{T}$ so that they meet their time and processor constraints. Depending on the computation environment, the scheduling algorithm may also try to optimize some performance metric (e.g., minimize average response time) while meeting these constraints. Obviously, this task system can be implemented on different system models including the time-shared computers model, the processor-pool model and the workstation model.

Two approaches can be taken to implement a VOC system. In the *limited-resource* approach, the processing power available in the system is less than the sum of the processing powers of all virtual computers. In the *plenty-resource* approach, the available processing power in the system is equal to or greater than the sum of the processing powers of all virtual computers. The limited-resource approach relies on the assumption that all users do not use their virtual computers simultaneously. At times, the system may not be able to guarantee everyone the promised service quality. The approach can be modeled by including in the performance metric the concept of meeting the promised service qualities. For example, minimizing total lateness can be selected as the performance metric. In the plenty-resource approach, if all the virtual computers are not fully utilized, there will be idle cyles in the system. The idle cycles can be used to maximize a performance metric. For example, this approach is suitable for owner-based systems in which idle cycles are shared among other users. This approach can be implemented on other system models as well. For example, if a time-shared computer system model is used, then each user can be allocated sufficient time-slices so that the user can get a performance that is equivalent to his virtual computer. In this approach, meeting

the promised service qualities can be viewed as a correctness criterion that must be preserved while optimizing another performance criterion.

The VOC paradigm models computing environments where each user wants to own a specific computer. It can also be used to model computing environments where the number of users who can access the system at a given time is fixed by limiting the access points to the system, and each access point is promised a given performance. In this case, each access point is a virtual computer which can be used by different users at different times (e.g., a lab consisting of n personal computers).

We note, however, that the VOC paradigm cannot be used to model a computing environment in which users are willing to accept any performance. For example, university students may have no alternative but to accept whatever level of performance is currently provided by the school's distributed computing environment, even if this means waiting 10 seconds for simple editor response during final project week. Similarly, the VOC paradigm is not appropriate for modeling hybrid computing environments in which there are both owners and "property-less" users who are willing to accept any quality of service. While it may be possible to extend the VOC paradigm to handle such environments, these issues are beyond the scope of this paper.

4 Model Assumptions

In this section, we introduce a number of simplifying assumptions which allow us to study scheduling in the VOC paradigm. These restrictions are not inherent to VOC and we expect to relax them in the future.

When we introduced the task model in Section 3, we did not consider possible dependencies among tasks. These dependencies can be specified by precedence constraints. In this paper, we assume that tasks are mutually independent. We also assume that tasks are preemptable and the cost of preemption is negligible. Further, we assume that tasks are computation intensive and do not require I/O. We denote the length of a task T_i by l_i, though we do not necessarily assume that the length of task is known when the task enters the system.

We only consider processing resources (CPUs). That is, we assume that the system always has sufficient amounts of other resources such as memory, secondary storage and network bandwidth. We measure the quality of a service using task response times. There are other aspects of service quality such as security and protection, but we will not cover them in this paper.

We assume that each virtual computer is of the same type; that is, they all have the same CPU type (speed) and local scheduling algorithm. We consider two local scheduling algorithms:

- *First Come First Served* (FCFS). A newly arrived task is inserted at the tail of the ready queue. The CPU is allocated to the task at the head of the ready queue until the task completes.
- *Round-Robin* (RR). A time slice is defined. A newly arrived task is inserted at the tail of the ready queue. The CPU is allocated to the task at the head of the ready queue for a time interval of up to one time slice. If the task has not completed, it is deleted from the head of the queue and inserted at the tail of the queue.

In this paper, we model the system as a set of equivalent processors. Because the processors on which the distributed operating system is built may not physically share memory, we must also account for migration cost when a task moves from one processor to another in such systems. In general, we say the migration time of task T_i is m_i seconds, of which $m_{i_{node}}$ seconds are the overhead on the computer from which the task is migrated. We assume that the cost of starting or restarting a task on any node is negligible.

5 Scheduling in Distributed and Real Time Systems

In this section, we highlight the main differences between scheduling in VOC systems and traditional scheduling in distributed and real time systems. Scheduling a set of tasks is sequencing the tasks on one or more processors in order to optimize a given criterion. In traditional general-purpose distributed operating systems, common performance criteria are throughput, average response time, and idle processor cycles. However, the notion of providing predictable performance to each user is not emphasized. A VOC system aims to provide each user with a service which is at least as good as that which would have been obtained by executing on the user's virtual computer. Depending on the computation environment, another performance metric may also be optimized without violating time constraints of tasks. Recall that the time constraint of a VOC task is defined as the response time on an imaginary computer (virtual computer).

Below, we present examples to demonstrate that algorithms that balance load, minimize average response time, or minimize idle cycles do not simultaneously satisfy the time constraints that the VOC paradigm imposes on tasks (despite the fact that there is a schedule that meets these

Virtual Computer	Task	a_i	l_i	d_i
V_1	T_1	0	1000	2998
V_1	T_2	0.2	1000	2999
V_1	T_3	0.4	1000	3000
V_2	T_4	0	1000	1000

Figure 1: Tasks characteristics in Example 1.

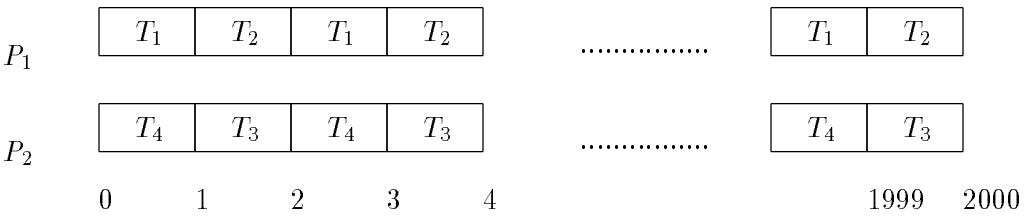


Figure 2: Schedule on processors P_1 and P_2 in Example 1.

constraints). In all the examples, there are two real processors (P_1 and P_2) and two virtual computers (V_1 and V_2). The processing speeds of real processors and virtual computers are identical. The virtual processors use an RR local scheduling scheme with time slice length equal to one unit of time.

Example 1: Consider the four tasks in Figure 1 where T_1, T_2 and T_3 are submitted to V_1 and T_4 is submitted to V_2. Figure 1 gives the task arrival times (a_i), lengths (l_i) and deadlines (d_i where $d_i = \bar{c}_i$). Suppose that system balances load and schedules the processors P_1 and P_2 in RR fashion with time slices of unit length. Figure 2 depicts the schedule on processors P_1 and P_2 that balances the load. As illustrated in Figure 2, task T_4 misses the deadline imposed by the VOC paradigm, completing at $t = 1999$ rather than $t = 1000$. □

Example 2: Consider now the four tasks with the characteristics described in Figure 3. Assume that the scheduling algorithm seeks to minimize the average response time. Figure 4 illustrates the schedule on processors P_1 and P_2 which yields the minimum average response time. T_4 completes at $t = 3001$, missing its deadline of $t = 2003$. Although the average response time is minimized, the time constraints are not met. □

We now consider scheduling techniques from owner-based systems that attempt to minimize idle processor cycles while respecting ownership rights. Two methods are used to maintain ownership

Virtual Computer	Task	a_i	l_i	d_i
V_1	T_1	0	1000	2998
V_1	T_2	1	1000	2999
V_1	T_3	2	1000	3000
V_2	T_4	3	2000	2003

Figure 3: Tasks characteristics in Example 2.

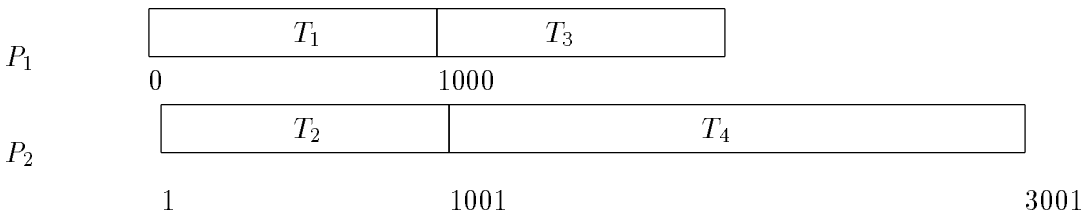


Figure 4: Schedule on processors P_1 and P_2 in Example 2.

rights. When an owner reclaims his computer, foreign tasks are either preempted as in [DO91, LLM88], or they are assigned a lower priority as in [KC91]. We give counterexamples to demonstrate that neither of these methods respects the time constraints imposed by the VOC paradigm.

Example 3: Let T_1 and T_2 be tasks submitted to V_1, and let T_3 be a task submitted to V_2. Further, assume that V_2 is not used until $t = 12$, at which time it is reclaimed. Figure 5 depicts the characteristics of the three tasks. Suppose that processors P_1 and P_2 belong to the owners of V_1 and V_2 respectively, and that the system migrates a foreign task when the owner reclaims his virtual computer. Migration of T_2 takes 10 time units and incurs 4 time units of overhead on the processor from which the task is migrated. The local scheduling algorithms of processors P_1 and P_2 are RR with the length of time slice being one unit of time. Figure 6 illustrates the schedule on processors P_1 and P_2. T_2 and T_3 complete at $t = 30$ and 116 respectively and miss their deadline. Hence, such systems are not VOC systems. □

Virtual Computer	Task	a_i	l_i	$m_{i_{node}}$	m_i	d_i
V_1	T_1	0	12			14
V_1	T_2	10	10	4	10	22
V_2	T_3	12	100			112

Figure 5: Tasks characteristics in Example 3.

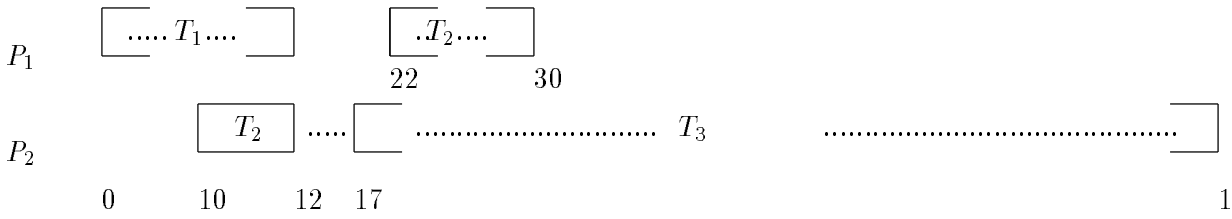


Figure 6: Schedule on processors P_1 and P_2 in Example 3.

Virtual Computer	Task	a_i	l_i	$m_{i_{node}}$	m_i	d_i
V_1	T_1	0	100			110
V_1	T_2	10	10	5	10	29
V_2	T_3	12	100			112

Figure 7: Tasks characteristics in Example 4.

Example 4: Let T_1 and T_2 be tasks submitted to V_1, and let T_3 be a task submitted to V_2. V_2 is available until $t = 12$, at which time it is reclaimed. Figure 7 depicts the characteristics of these three tasks. Suppose that processors P_1 and P_2 belong to the owners of V_1 and V_2 respectively, and that the system preempts a foreign task gives a lower priority to the task when the owner reclaims his virtual computer. The lower priority tasks are executed whenever there is no higher priority task to execute. Figure 8 illustrates the schedule on processors P_1 and P_2. T_2 completes at $t = 120$ and misses its deadline. Hence, such systems are not VOC systems. □

Despite the "deadline" oriented nature of scheduling in both VOC and real-time (RT) systems, scheduling algorithms for RT systems are not directly applicable to VOC systems. This is because RT systems assume that the characteristics of a task, which include the length of the task and

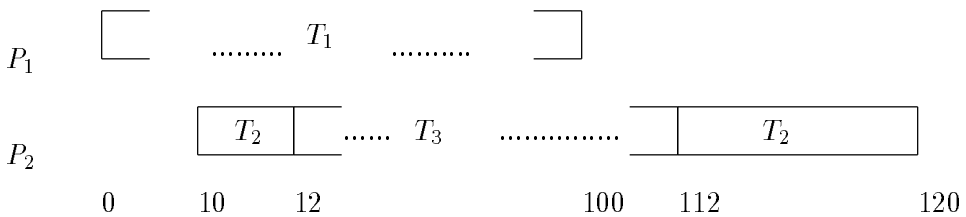


Figure 8: Schedule on processors P_1 and P_2 in Example 4.

the deadline of the task, are known to the scheduler when the task arrives at the system. In RT systems, tasks are assigned fixed deadlines according to some criteria. This is in contrast to VOC systems where a task has an unpredictable time constraint. The deadline of a task depends on the length of the task, CPU speed, the local scheduling algorithm, and the load of the virtual computer. Since the load changes dynamically, the deadline of a task is not fixed at the arrival time, even if the length of the task is known. Furthermore, in VOC systems, the length of a task is not necessarily known to the scheduler when the task enters the system, whereas RT systems are based on the assumption that the length of each task is known a priori. In RT systems, the length of a task is typically assumed to be equal to its worst case computation time. Since the VOC paradigm targets general purpose computing environments, where the length of a task can be much shorter than its worst case computation time, it is impractical to make that assumption for VOC systems.

Some of the typical scheduling criteria in RT systems can be used for some VOC systems depending on the computation environment. Typical criteria in RT systems are minimizing the number of tardy tasks, minimizing the average tardiness, and maximizing the system value [SR88]. In addition, static scheduling algorithms from RT systems can act as a good basis of comparison for some of the scheduling algorithms for VOC systems.

6 Scheduling in VOC Systems

In this section, we examine the basic scheduling problems implicit in the VOC paradigm. We first study the existence/nonexistence of optimal scheduling algorithms for single and multiprocessor systems and then consider appropriate VOC performance metrics.

Under VOC, the goal is to optimize a given criterion without undermining any individual user's performance; that is, without delaying any task in the system beyond the time when it would have been completed if the task were executed on the virtual computer. The selection of the performance metric depends on the computation environment and the implementation of the VOC paradigm.

Let us first consider the case where the sole goal is meeting deadlines in a limited-resource environment. A schedule for the set of tasks $\mathcal{T}$ is called *feasible* if each task in $\mathcal{T}$ is completed by its deadline in the schedule. A scheduling algorithm is said to be *optimal* if, for any set of tasks, it always produces a feasible schedule when one exists. A basic question to ask is whether there is an optimal dynamic scheduling algorithm.

Virtual Computer	Task	a_i	l_i	d_i
V_1	T_1	0	10	10
V_1	T_2	4	8	18

Figure 9: Tasks characteristics in Example 5.

A dynamic scheduler does not know about the characteristics of a task a priori. Rather, it obtains this information only when the task enters the system. The following characteristics of a task T_i are available to the scheduler at time $t \geq a_i$: The arrival time a_i of the task; the number of time units that T_i has executed in the system until t, denoted by $l_i(t)$; and the time at which $l_i(t)$ units of T_i would be executed if T_i were executed on the virtual computer, denoted by $\bar{t}_i$. Using $\bar{t}_i$, we can derive the number of time units by which the task precedes its imaginary execution on the virtual computer, denoted by $p_i(t)$, where $p_i(t) = \bar{t}_i - t$. The following example illustrates these definitions.

Example 5: Consider a system consisting of two processors P_1 and P_2, and a single virtual computer V_2. Tasks T_1 and T_2 are submitted to virtual computer V_1 with FCFS as the local scheduling algorithm. Figure 9 depicts the tasks' attributes, and Figure 10 illustrates a possible schedule of these tasks. At time $t = 1$, the scheduler knows that $a_1 = 0$, $l_1(1) = 1$, $\bar{t}_1 = 1$ and $p_1(1) = 0$. At time $t = 5$, the scheduler knows that $a_1 = 0$, $l_1(5) = 5$, $\bar{t}_1 = 5$, $p_1(5) = 0$ and $a_2 = 4$, $l_2(5) = 1$, $\bar{t}_2 \geq 6$, $p_2(5) \geq 1$. At time $t = 11$, the scheduler knows and $a_2 = 4$, $l_2(11) = 7$, $\bar{t}_2 = 17$ $p_2(11) = 6$. □

We now present some basic results concerning scheduling in a uniprocessor VOC system where the real processor and the CPUs of the virtual computers have identical speed.

Theorem 1: Consider a limited-resource uniprocessor VOC systems where the processing speed of the processor is identical to that of the virtual computers. If the local scheduling algorithms of the virtual computers are FCFS and RR respectively, then the FCFS and RR schemes are optimal.

Proof: There is a feasible schedule, if and only if the bursts of any two virtual computer never overlap. Given this observation, if the real processor uses the same FCFS or RR scheduling algorithm as the virtual processor, it will always meet the VOC deadline. Hence, the algorithm is optimal. □

Now, we consider uniprocessor systems where the processing speed of the processor is k times

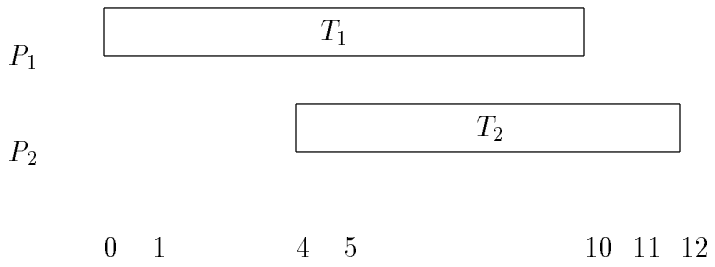


Figure 10: Schedule on processors P_1 and P_2 in Example 5.

faster than the processing speed of the CPUs of the virtual computers. This means that execution of a task on a computer that is equivalent to the virtual computer will take k times longer than it requires on the real processor.

Theorem 2: Consider a limited-resource uniprocessor VOC systems where the real processor is faster than the virtual computers. If the length of tasks are not known, then no optimal scheduling algorithm exists.

Proof: The proof relies on the fact that for any scheduling algorithm which sequences tasks in a given order without knowledge of task lengths, one can find a task set for which the algorithm fails. Consider a uniprocessor system where the processor is $k = 2$ times faster than the CPUs of virtual computers and assume the task set in Figure 11. Note that the nature of the local scheduling algorithm of the virtual computers does not affect the example, since only one task is submitted to each virtual computer. There are six possible ways of sequencing these tasks on the uniprocessor. The sequence depicted in Figure 12 is in fact a feasible schedule. However, since the scheduler does not know the length of tasks, all tasks appear to be the same to the scheduler. Thus, if a scheduler algorithm executes tasks of virtual computers in a given order, we can always find another set of tasks for which the algorithm fails. The proof is valid for any $k > 1$ and any number of virtual computers. □

Once the lengths of tasks are known, the scheduler can also calculate the deadline for task T_i at time $t \geq a_i$ denoted by $d_i(t)$. If the virtual computers use the FCFS scheme as the local scheduling algorithm, then $d_i(t) = d_i$ for $t \geq a_i$. If the virtual computers use the RR scheme as the local scheduling algorithm, then $d_i(t)$ is a monotonically increasing function and is always smaller than d_i. For any two tasks T_i and T_j of the same virtual computer, if $d_i(t_1) < d_j(t_1)$ holds, then $d_i(t_2) < d_j(t_2)$ holds for any $t_2 \geq t_1$, whereas for two tasks T_i and T_j from different virtual

Virtual Computer	Task	a_i	l_i	d_i
V_1	T_1	0	2	4
V_2	T_2	0	0.5	1
V_3	T_3	0	1	2

Figure 11: Tasks characteristics in Theorem 2.

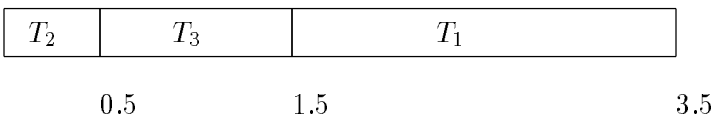


Figure 12: A feasible schedule in Theorem 2.

Virtual Computer	Task	a_i	l_i	d_i
V_1	T_1	0	2	4
V_2	T_2	0	2	8
V_2	T_3	1	2	8
V_3	T_4	2	0.5	3
V_4	T_5	2	0.5	3
V_5	T_6	2	0.5	3
V_6	T_7	2	0.5	3

Figure 13: Tasks characteristics in Theorem 3.

computers, this is not necessarily the case. If the lengths of tasks are known, the earliest deadline algorithm [SR88] is optimal for uniprocessor VOC systems where the virtual computers use the FCFS scheme as the local scheduling algorithm. However, this is not the case for uniprocessor VOC systems where the virtual computers use the RR scheme as the local scheduling algorithm.

Theorem 3: Consider a limited-resource uniprocessor VOC systems where processing speed of the processor is greater than that of one of the virtual computers with RR scheme as the local scheduling algorithm. If the arrival times of tasks are not known, then no optimal scheduling algorithm exists.

Proof: The proof relies on the fact that for any scheduling algorithm which sequences tasks in a given order without the knowledge of arrival times of tasks, one can find a task set for which the algorithm fails. Consider a uniprocessor system where the processor is $k = 2$ times faster than the CPUs of virtual computers and virtual computers with RR local scheduling algorithm with time slice length being one unit of time. Suppose that at time $t = 0$ tasks T_1 and T_2 are submitted to virtual computers V_1 and V_2 respectively, where $l_1 = l_2 = 2$, and the next task arrives at the system at $t = 1$. There are three possible ways of sequencing T_1 and T_2 in the time interval $[0, 1)$—executing T_1 in this interval, executing T_2 in this interval, or executing both T_1 and T_2 for some fraction of the time in this interval. The first sequencing order fails for the task set in Figure 13 (see Figure 14, T_1 misses its deadline), the second sequencing order fails for the task set in Figure 15 (see Figure 16, T_2 misses its deadline), whereas the third sequencing order fails with both task sets, although there are feasible schedules for both task sets. The proof is valid for any $k > 2$ and any number of virtual computers. □

Let us now consider multiprocessor VOC systems which use the limited-resource approach. We restrict the model so that processors and virtual computers have identical processing speed, tasks have no processor constraints and migration costs are negligible.

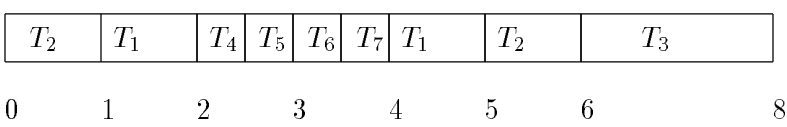


Figure 14: First sequencing order in Theorem 3.

Virtual Computer	Task	a_i	l_i	d_i
V_1	T_1	0	2	8
V_1	T_8	1	2	8
V_2	T_2	0	2	4
V_3	T_4	2	0.5	3
V_4	T_5	2	0.5	3
V_5	T_6	2	0.5	3
V_6	T_7	2	0.5	3

Figure 15: Tasks characteristics in Theorem 3.

Theorem 4: Consider a limited-resource multiprocessor VOC systems. If the arrival times of tasks are not known a priori, then no optimal scheduling algorithm exists.

Proof: It has been proven for RT systems that there cannot be an optimal scheduling algorithm for multiprocessor systems if arrival times of tasks are not known a priori [SR88]. We prove that this remains the case in VOC where the time constraint of a task varies as a function of both its length and the load and local scheduling algorithm of the virtual computer. To this end, we show that any possible scheduling decision at a given time t fails if the arrival times of the tasks that enter the system after t are not known at t (although there is a feasible schedule). Consider the following example. Let V_1, V_2, V_3 and V_4 be virtual computers which use as the local scheduling algorithm an RR scheme with time slice length equal to one time unit. Let T_1 and T_2 be tasks submitted to V_1 and tasks T_3 and T_4 be tasks submitted to V_2. Figure 17 depicts the characteristics of the tasks. At time $t = 10$, the scheduler can choose any of the following pairs of tasks to execute: T_2 and T_3, or T_2 and T_4, or T_3 and T_4. Figure 18 depicts the case when the scheduler selects T_2 and T_3. However, at time $t = 11$, tasks T_5 and T_6 arrive at virtual computers V_3 and V_4 respectively, where $l_5 = l_6 = 1$ and $d_5 = d_6 = 12$. T_4 misses its deadline. On the other hand, Figure 19 depicts the case when the scheduler selects T_3 and T_4. At time $t = 11$, T_2 is scheduled again. However, at time $t = 12$, tasks T_5 and T_6 arrive at virtual computers V_3 and V_4 respectively, where $l_5 = l_6 = 2$

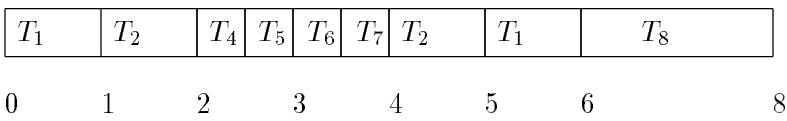


Figure 16: Second sequencing order in Theorem 3.

Virtual Computer	Task	a_i	l_i	d_i
V_1	T_1	0	10	12
V_1	T_2	8	4	14
V_2	T_3	10	1	11
V_2	T_4	10	1	12

Figure 17: Tasks characteristics in Theorem 4.

and $d_5 = d_6 = 13$. T_2 misses its deadline. Furthermore, if the scheduler selects T_2 and T_4 at $t = 10$ for execution, T_3 will miss its deadline. If FCFS scheme is used as the local scheduling algorithm for virtual computers, the task set in Figure 20 can be used with the same example to show that there can be no optimal scheduling algorithm for multiprocessor systems. □

In VOC systems based on the plenty-resource approach there always exists a feasible schedule for any set of tasks. For instance, if the number of processors in the system is equal to the number of virtual computers, and the processors and virtual computers have identical processing speed, then the feasible schedule can be obtained by simply assigning each virtual computer a processor, and executing all tasks of the virtual computer on the same processor.

In VOC systems based on the plenty-resource approach, the objective is to optimize a performance metric (e.g., maximize utilization, minimize average response time) while meeting deadlines. If the meeting of timing constraints is treated as a correctness criterion, then the scheduling algorithm must choose from among the feasible schedules the one that is optimal with respect to a given performance criterion. We refer to such a scheduling algorithm as an *optimal scheduling algorithm*, and to such a schedule as an *optimal schedule*. (Note that the definition of an optimal scheduling algorithm plenty-resource approach is different than the one for the limited-resource approach.)

Now, we examine the scheduling problem in multiprocessor VOC systems where the number of processors is equal to the number of virtual computers, and where the real processors and

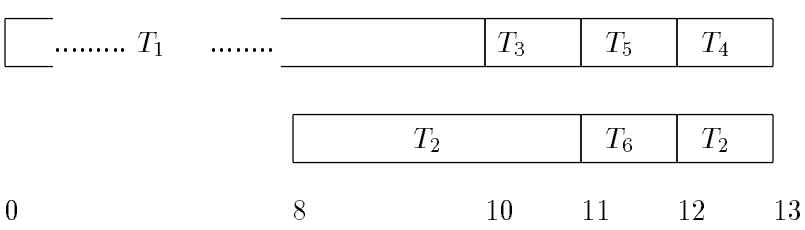


Figure 18: Task scheduling.

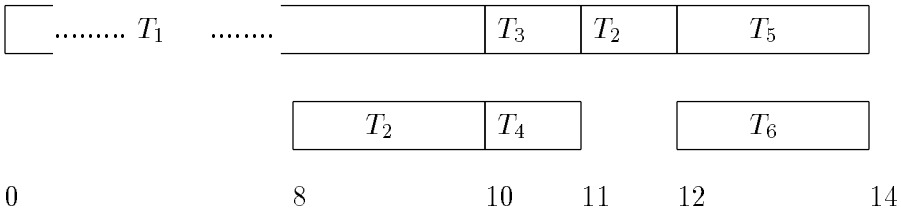


Figure 19: Task scheduling.

the virtual computers have identical processing speed. We first assume that migration costs of tasks are negligible and consider minimization of average response time as the performance goal. Obviously, for the plenty-resource approach, if the lengths of tasks are not known a priori, there exists no optimal scheduling algorithm that minimizes average response time without violating timing constraints. However, if the lengths of tasks are known, the algorithm below can be used to minimize average response time without violating timing constraints.

The scheduler maintains a list that contains $d_i(t) - (l_i - l_i(t))$ for each task T_i that is not completed. Three events trigger the scheduling— arrival of a new task, completion of a task, and completion of a schedule interval. A schedule interval is the interval for which the scheduler assigns a processor to a task. At each event, the scheduler updates the list; assigns each task T_i for which $d_i(t) - (l_i - l_i(t)) = t$ holds to a processor; assigns the remaining processors to tasks whose $l_i - l_i(t)$ is shortest for a time interval $[t, t')$, where t' is the earliest time at which $d_i(t') - (l_i - l_i(t')) = t'$ will hold for any of the unscheduled tasks. If the virtual computers use FCFS local scheduling algorithm, then $d_i(t) = d_i$ for each task T_i in the algorithm above.

If the migration costs of tasks are not negligible, the the following holds.

Theorem 5: Consider VOC systems based on the plenty-resource approach where migration costs of tasks are not negligible. If arrival times of tasks are not known a priori, then there exists

Virtual Computer	Task	a_i	l_i	d_i
V_1	T_1	0	10	10
V_1	T_2	8	4	14
V_2	T_3	10	1	11
V_2	T_4	10	1	12

Figure 20: Tasks characteristics in Theorem 4.

no optimal scheduling algorithm that minimizes average response time without violating timing constraints.

Proof: Consider a two processor system (P_1 and P_2) with two virtual computers (V_1 and V_2) that use FCFS as the local scheduling algorithm. Consider the task set illustrated in Figure 21. In order to meet deadlines, T_1 and T_3 must execute starting at their arrival times without interruption until completion. Depending on the values of the variables a_2, a_3, and l_3, one of the following three schedules for T_2 will yield minimum average response time while meeting its deadline. First, T_2 is executed on P_2 until $t = a_3 - m_{2_{node}}$ and then migrated to P_1. Its response time will be

$$\max\{10, (a_3 + m_2)\} + 10 - l_2(a_3) + m_{2_{node}}.$$

Second, T_2 is executed on P_2 until $t = a_3$ and then preempted until T_3 completes and continued on P_2. In this case, its response time will be

$$a_3 + l_3 + 10 - l_2(a_3).$$

Third, T_2 is executed on P_1 after T_1 is completed, in which case its response time will be 20. In order to make the optimal decision, arrival times must be known by time $t = a_3 - m_{2_{node}}$. (Note that T_2 can be aborted on P_2 and restarted on P_1 without cost). But, by appropriately selecting the values of the free variables, we can always find a task set such that the scheduler will not know the arrival times by time $t = a_3 - m_{2_{node}}$ This argument is valid for any number of processors. One can give a similar example for virtual computers that use RR as the local scheduling algorithm. □

In a general purpose computing environment, it is not realistic to assume a priori knowledge of tasks' arrival times. Therefore, one must devise heuristic scheduling algorithms for VOC systems using known characteristics of tasks; namely, a_i, $p_i(t)$, $l_i(t)$ and m_i. In the case that l_i is known, we can also use $d_i(t)$ and l_i. The static scheduling algorithms for RT systems can be used to compare the heuristic scheduling for VOC systems depending on the performance metric.

Virtual Computer	Task	a_i	l_i	d_i
V_1	T_1	0	10	10
V_1	T_2	$0 < a_2 < a_3$	10	20
V_2	T_3	$a_3 < 10$	l_3	$a_3 + l_3$

Figure 21: Tasks characteristics in Theorem 5.

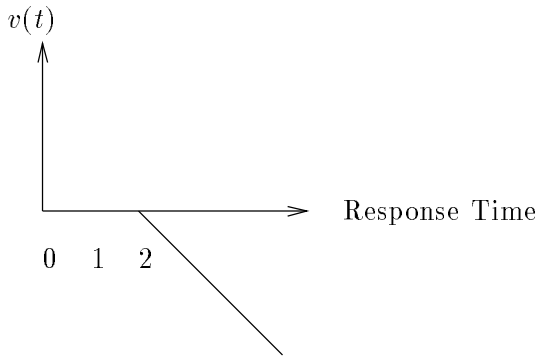


Figure 22: The value function of task with $\bar{c} = 2$.

Appropriate performance metrics depend on the computation environment. For VOC systems, performance metrics should include the concept of meeting VOC time constraints. In particular, value functions provide an appropriate mechanism to describe criterion such as minimizing lateness of tasks, or reducing the average response time while preserving deadlines. Different value functions can be defined depending on the importance of meeting timing constraints. For example, in order to capture minimization of lateness, each task T_i can be assigned value function $v_i(t)$, where:

$$v_i(t) = \begin{cases} 0 & t \leq \bar{c}_i \\ \bar{c}_i - t & t > \bar{c}_i \end{cases}$$

Figure 22 depicts the value function of task T_i with $\bar{c}_i = 2$. Maximizing

$$\sum_i v_i(c_i)$$

is equivalent to minimizing lateness.

On the other hand, minimization of average response time while satisfying the time constraints of each task can be approximated with the following value function:

$$v_i(t) = \begin{cases} \bar{c}_i - t & a_i \leq t \leq \bar{c}_i \\ (\bar{c}_i - t)^3 & t > \bar{c}_i \end{cases}$$

Figure 23 depicts the value function of a task which arrives at $t = 0$ and whose deadline is at $t = 10$. The value function decreases linearly until to the imaginary completion time $\bar{c}_i$ of task T_i on the virtual computer and then drops abruptly. The objective of the scheduling is to maximize

$$\sum_i v_i(c_i).$$

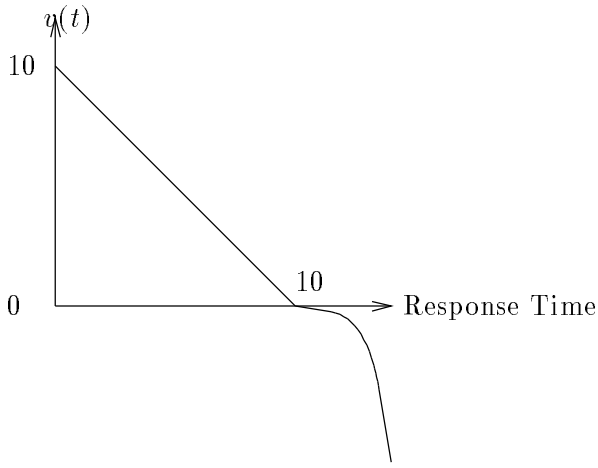


Figure 23: The value function of a task with $a = 0$ and $\bar{c} = 10$.

In summary, this section presented a study of the basic scheduling problems for VOC systems. We established whether optimal scheduling algorithms exist for single and multiprocessor VOC systems, and proposed the value function approach for specification of performance metrics for VOC systems. Given this framework, heuristics can be developed that select schedules with good performance metric values.

7 The Eagle Distributed Operating System

The VOC paradigm is currently being implemented in the Eagle distributed operating system project at the University of Texas at Austin. The Eagle system is a collection of heterogeneous workstations and resources owned by individuals. The owners of resources specify the policies that govern the conditions under which their workstations can be shared, and the load sharing algorithm guarantees a service to each user at least as good as if the resources of the user are shared with respect to these policies.

One of the policies deals with reclamation of virtual computers. In the Eagle system, a virtual computer is reclaimed if its owner demands service which is at least as good as the one that could have been obtained if the owner were executing his tasks on his virtual computer. Reclamation can be either implicit or explicit. In the first case, the user specifies the conditions under which his virtual computer must be reclaimed, and the system reclaims the virtual computer when these conditions hold. In the second case, the user explicitly reclaims his virtual computer through a command. If the user always requires the service quality promised to him (as in this paper), the

user specifies that his virtual computer must be reclaimed whenever he submits tasks to the system after an idle interval. Extending the definition of reclamation increases the number of processing cycles that can be shared while supporting predictability when it is desired.

Various heuristic algorithms are studied for processor scheduling in the Eagle system. The heuristics are based on the the known characteristics of tasks, namely, a_i, $p_i(t)$, $l_i(t)$ and m_i. In the case that l_i is known, we also use $d_i(t)$ and l_i. The Eagle system incorporates two useful extensions to the VOC paradigm. First, the Eagle system can deal with tasks for which $d_i > \bar{c}_i$ holds. This extension is useful when modeling a task that must be completed before a deadline, but c_i can be later than $\bar{c}_i$. For example, a user may submit a two hours long task before going home at 6pm with the requirement that the result be computed by 8am next morning, when he arrives at his office. This extension is useful to utilize processing cycles for more critical tasks and increase the performance metric.

Second, the Eagle system can deal with tasks for which $d_i < \bar{c}_i$ holds. This extension is useful to model a task that must be completed before a deadline, which is earlier than its completion time on the virtual computer. For example, a user has a paper submission deadline at 6pm, whereas he submits two three hours long simulation tasks at 3pm. He may ask his friends to designate their machines to him to meet the paper deadline.

8 Conclusions

We introduced the virtually owned computers (VOC) paradigm which incorporates the predictability and choice properties into the design of distributed operating systems. We examined the basics of the VOC scheduling problem and concluded that heuristic approaches need to be developed. While initial experience with the Eagle project is encouraging, the design and evaluation of the VOC paradigm on various system models presents numerous challenging problems. Examples are scheduling under precedence constraints, scheduling I/O bound tasks, development of adequate performance metrics, development of efficient scheduling algorithms for various system models, scheduling with partial state information, and scheduling when virtual computers are heterogeneous and when the system consists of heterogeneous resources. In addition, naming and protection must also be reconsidered in the light of VOC paradigm.

References

[BL85] A. Barak and A. Litman. MOS: A multicomputer distributed operating system. *Software Practice and Experience*, 15(8):725–737, August 1985.

[BOS93] A. Goldberg B. Ozden and A. Silberschatz. Scalable and non-intrusive load sharing in owner-based distributed systems. In *Proceedings of 5th IEEE Symposium on Parallel and Distributed Processing*, Dallas, Texas, December 1993.

[DO91] F. Douglis and J. Ousterhout. Transparent process migration: Design alternatives and the sprite implementation. *Software–Practice and Experience*, 21(8):757–85, August 1991.

[KC91] P. Krueger and R. Chawla. The Stealth distributed scheduler. In *Proceedings of 11th International Conference on Distributed Computing*, pages 336–343, Los Alamitos, California, 1991.

[LLM88] M. J. Litzkow, M. Livny, and M. W. Mutka. A hunter of idle workstations. In *Proceedings of 8th International Conference on Distributed Computing*, pages 104–111, Los Alamitos, California, 1988.

[NH82] R. M. Needham and A. J. Herbert. *The Camridge Distributed Computing System.* Addison-Wesley, Reading, Mass, 1982.

[RPT90] K. Thompson R. Pike, D. Presotto and H. Trickey. Plan 9 from Bell Labs. In *Proceedings of the Summer 1990 UKUUG Conference*, pages 1–9, London, July 1990.

[SR88] J. A. Stankovic and K. Ramamritham. *Tutorial: Hard Real-Time Systems.* IEEE Computer Society Press, 1988.

[Stu88] M. Stumm. The design and implementation of a decentralized scheduling facility for a workstation cluster. In *Proceedings of 2nd Conference on Computer Workstations*, pages 12–22, Los Alamitos, California, 1988.

[Tan86] S. J. Mullender A. S. Tannenbaum. The design of a capability based operating system. *Computer Journal*, 29(4), 1986.

[TR85] A. S. Tannenbaum and R. Van Rennesse. Distributed operating systems. *ACM Computing Surveys*, 17(4):419–470, November 1985.