Future Generation Computer Systems 29 (2013) 61–73

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Deadline prediction scheduling based on benefits

Javier Palanca *, Marti Navarro, Ana García-Fornes, Vicente Julian

DSIC - Universitat Politècnica de València, Camino de Vera s/n, 46022, Valencia, Spain

ARTICLE INFO

Article history:
Received 4 April 2012
Received in revised form
7 May 2012
Accepted 17 May 2012
Available online 27 May 2012

Keywords:
Guaranteed processor
Service-oriented computing
Scheduling

ABSTRACT

This paper describes a scheduling algorithm that composes a scheduling plan which is able to predict the completion time of the arriving tasks. This is done by performing CPU booking. This prediction is used to establish a temporal commitment with the client that invokes the execution of the task. This kind of scheduler is very useful in scenarios where Service-Oriented Computing is deployed and the execution time is used as a parameter for QoS. This scheduler is part of an architecture that is based on the Distributed Goal-Oriented Computing paradigm, which allows agents to express their own goals and to reach them by means of service compositions. Moreover, the scheduler is also able to prioritize those tasks which provide greater benefits to the OS. In this work, the scheduler has been designed in several iterations and tested by means of a set of experiments that compare the scheduler algorithm with a representative set of scheduling algorithms.

1. Introduction

Nowadays, service-oriented applications are a new way of developing flexible and distributed solutions. In applications of this kind, the Quality of Service (QoS) has become one of the highest priorities for service providers and their clients. Due to the dynamic and unpredictable nature of the web, providing an acceptable QoS is quite a challenging task. Service consumers need guarantees that the services will be executed with a minimum level of quality. For this reason, service providers must offer and fulfill this commitment to quality.

Service execution time is one of the most important QoS parameters in web services. There are some systems, such as the RT-MOVICAB-IDS system [1], where the execution of a service on time is considered to be a critical parameter. Therefore, it is necessary to give a valid response before a certain instant in time. Otherwise, the system may become inefficient.

Time as a QoS parameter is not only important when executing a single service but also when a set of services is executed as part of a service composition. In this case, the problem resolution requires the collaboration of several services and, therefore, both the execution time of each individual service and the execution time of the service composition that fulfills the goal must be taken into account.

There are several proposals that introduce the time parameter into web services as a form of expressing temporal constraints. Some of these proposals are shown in [2–4]. Moreover, there are other proposals such as [5–7] that use the time parameter to guide the service composition. In all these proposals, the time parameter is considered from the point of view of the descriptive level, but at the execution level it provides no guarantee that these time constraints will be fulfilled. Moreover, the service execution time is often not known. This is because this time parameter may depend on many factors that are not controlled by the service provider, such as the system workload where the service is executed or the ending of other services that were previously necessary to execute the service. In these cases, the system must be able to diagnose how long it takes to complete each service. This is a very hard and complex process to procure with current architectures, since they are not focused on time predictability.

Therefore, it is very important to be able to determine the completion time of a single service or a service composition taking into account how the system workload is and, based on this, the system must be able to predict when the service or the composition ends. With this information, the system could become more efficient and could establish a commitment with the service client indicating when it is going to fulfill the service goal. When the system executes a service on time, it may gain a benefit offered by the service client. Moreover, the more quickly the services are fulfilled, the greater benefit the system gains. Thus, it would be desirable to have mechanisms that help to fulfill all services on time with the highest possible benefit.

In previous works, we presented an operating system (OS) architecture [8] that has the ability to control the service execution and also the ability to build service compositions taking into account the temporal constraints that the services have. This proposal increased the abstraction level provided by the operating system and their services. This allows us to offer an OS execution

* Corresponding author.
E-mail address: jpalanca@dsic.upv.es (J. Palanca).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.05.007

layer that is integrated into the network and also to offer security and reliability mechanisms, which are not available in lower levels of the OS architectures. Our OS architecture is based on a new paradigm called *Distributed Goal-Oriented Computing* [9]. This approach is based on Service-Oriented Computing concepts. Its purpose is to find solutions to problems through composition and execution of various services offered by different agents, taking the goal to be achieved as a starting point. This Distributed Goal-Oriented Computing paradigm suggests that agents are the components that provide services in a ubiquitous environment where users express their goals. Thus, users can reach a solution by finding a plan that achieves the selected goal with little user interaction.

The OS needs a scheduler to establish a proper commitment that guarantees that services end at a time agreed upon by both the client and the provider. This module is an important part of the OS since it is responsible for distributing the CPU time among all of the services in execution. The scheduler distributes the CPU time by means of a scheduling algorithm. To do this, the scheduling algorithm plans the order of the execution of the services that were invoked by the distributed environment. There are many schedulers used for general purpose operating systems. These approaches, as discussed below, do not satisfy all the needs of the Distributed Goal-Oriented Computing paradigm. Thus, in this paper, we propose new scheduling algorithms that are not only able to schedule tasks from a distributed environment [10], but they are also able to analyze when the service ends its execution and to plan the delivery of services with the intention of maximizing the benefit obtained by the OS. In this paper, we also compare the proposed scheduling algorithms with other classical scheduling algorithms.

The paper is organized into the following sections: Section 2 describes the state of the art for scheduling algorithms. Section 3 presents a new scheduling algorithm based on planning-based scheduling. Section 4 presents a set of experiments to compare and refine the different algorithms presented in this paper. Finally, Section 5 presents our conclusions.

2. Scheduling algorithms

As started above, this paper presents a new scheduling algorithm that allows the OS to make a scheduling plan that takes into account the prediction of when a task is going to finish its execution and how much benefit (in quantitative terms) this execution will bring to the OS. This section explores a set of representative schedulers that use different techniques to share the processor (fair algorithms, real-time algorithms, or non-preemptive algorithms). In this section, we also explore a category of schedulers called Planning-based scheduling which are closely related to the scheduler presented in this work.

General purpose Operating Systems schedulers usually use fair algorithms that equally share the processor time among all of the tasks. The most representative example is the Round Robin scheduling algorithm. This algorithm divides the processor time into units called *quantums* and gives each task the same number of quantums. When a task is running, it consumes its quantums; when it runs out of time quantums, the task is expelled from the processor in order to let another task with remaining quantums enter. This algorithm shares the same amount of time with all the tasks, so the feeling of interactivity is very high. However, this kind of scheduler algorithm does not allow the time when a task ends to be predicted because it is not possible to predict how many tasks the processor resources must be shared with.

There has been an interesting evolution of scheduling algorithms that try to share the processor time fairly. There is an algorithm called the *Completely Fair Scheduler* (CFS) by [11]. This algorithm was designed as a replacement of the $O(1)$ algorithm in the Linux kernel. Instead of using queues, this algorithm uses a complex structure called red-black trees. Red-black trees have a time complexity of $O(\log_n)$ for insert, search, and delete operations. From the red black tree, CFS efficiently picks the process that has used the least amount of time (this process is stored in the leftmost node of the tree). Even through this is a very fair scheduling algorithm that boosts multi-tasking performance, it is unable to predict when the tasks are going to finish.

There exists a very simple algorithm called FCFS (First-Come First-Served), which serves the tasks in order of arrival. This algorithm was very common in batch systems and implemented a queue that held the tasks in the order they came in. It is a non-preemptive algorithm, which means that until the active task is not finished it will not be interrupted. Of course, with this kind of algorithm, it is easy to make a prediction of the deadline because, in the worst case, it will be its WCET,[1] since it is certain that the task will not be interrupted. Its main drawback is that it is a non-preemptive algorithm. In a world where multi-tasking and interactivity are not just features but are requirements, this is not acceptable for distributed systems, which is a focus.

However, predicting the deadline of a task is also a different problem than those found in real-time problems. Real-time systems have to schedule tasks that must be run before an established instant of time, which is also called *deadline*. This deadline is mandatory in hard real-time systems, although deadline fails are not critical in soft real-time systems. These real-time systems use specific schedulers that ensure that every accepted task will be run before its deadline. For this reason, it is very important to have an accurately calculated WCET.

The most simple way of scheduling real-time tasks is to use a *cyclic executive*. This executive is a way of scheduling tasks in a real-time system that using cyclic tasks. These tasks have a period and a WCET, so the cyclic executive just has to create a fixed execution plan provided by the system designer. The cyclic executive can replace the whole OS since it only takes an infinite loop to run the tasks with the order established by the designer.

Planning-based scheduling is a kind of real-time dynamic scheduling that gives assurances to arriving jobs by implementing admission controls. These assurances are related to the ability of the system to meet the time constraints (deadlines) of the incoming tasks.

Planning-based scheduling (PBS) usually involves making a plan to run all the enqueued tasks, which implies assigning priorities to the tasks. When dynamic priorities are used, the relative priorities of tasks can change as time progresses and also when tasks are executed.

In general, PBS has to go through three steps: Feasibility analysis, Schedule construction, and Dispatching. The feasibility analysis is done to check the schedulability of a task (i.e., whether or not the time constraints of the task can be satisfied). This feasibility test is usually done when the task arrives to the system. These tests are more suited for periodic tasks since they have a periodic activation and the resources they need can be easily calculated and reserved. In planning-based approaches, this test is also applied to aperiodic tasks.

The schedule construction is the process of ordering the tasks to be executed. This order is stored for use in the dispatch step. This schedule construction is usually done when dynamic priorities are assigned. Finally, the dispatch step is in charge of deciding which tasks to execute next. This dispatch process may be to follow the established plan, depending on whether the system is preemptive

[1] Worst-case execution time: the maximum length of time a task could take to execute on a specific hardware platform [12].