Future Generation Computer Systems 28 (2012) 871–880

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

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

Evaluation of hierarchical desktop grid scheduling algorithms☆

Z. Farkas *, P. Kacsuk

MTA SZTAKI, 1518 Budapest P.O. Box 63., Hungary

ARTICLE INFO

Article history:
Received 20 May 2010
Received in revised form
24 November 2010
Accepted 28 December 2010
Available online 4 January 2011

Keywords:
Desktop grid
Scheduling
Volunteer computing
Model
Simulation

ABSTRACT

Desktop grids as opposed to service grids are the cost-effective way to gather large amount of volunteer computing resources for solving scientific problems. It is possible to create desktop grids with the help of only one single machine, where volunteer desktops will connect to process work. MTA SZTAKI has created the hierarchical desktop grid concept, where not only single computers, but also desktop grids may join other systems increasing their performance significantly. In the paper we investigate scheduling issues of hierarchical desktop grid systems, present scheduling algorithms focusing on different properties, and compare them using HierDGSim, the Hierarchical Desktop Grid Simulator.

1. Introduction

Desktop grids are the cost-effective way of gathering a large amount of computing resources from the operator's point of view: a small set (even one machine is feasible) of central services is used to store the applications and their workunits to process and compute resources offered by volunteers, who can join by installing a client application and registering in the central services (project servers). Afterwards, volunteer clients periodically communicate with the project servers to fetch work, process them, and upload the results back to the servers. Attractive scientific projects may have connected clients of the order of one million, for example SETI@home [1] has over 2, and Einstein@Home [2] has over one million machines connected according to Boinstats [3]. The work performed is rewarded by credits, a virtual measurement of how much work the given client has performed in favor of the project.

It follows from the nature of desktop grids that attached clients are likely to behave unexpectedly: they may produce false results due to CPU, memory or any other hardware problems, or they may want to achieve higher credits without actually performing the computation. In order to filter out such behavior, desktop grids use redundant computing; a given piece of work is computed by a given number of clients, and if a given amount of reported results are equivalent, the credit is granted to those clients that have reported the correct result.

Access to the desktop grid project servers is limited a small set of scientists and administrators, thus applications run by desktop grids is limited to these persons' needs.

Examples for desktop grid implementations are BOINC [4], XtremWeb [5] or OurGrid [6].

1.1. Hierarchical desktop grid concept

A natural extension of a given desktop grid's (e.g. *A*) performance is the addition of new client machines. This might be inconvenient, as each client may have to register in the given desktop grid. In the case of a heterogenous infrastructure and a big number of machines this requires notable effort from the infrastructure administrators. However, if the new attaching clients already take part in a desktop grid project (e.g. *B*), it would be practical to connect this desktop grid (*B*) to the one we would like to increase the performance of (*A*). This scenario is a simple use-case of a hierarchical desktop grid system, where a desktop grid (*B*) is processing workunits of another desktop grid (*A*).

SZTAKI Desktop Grid created by SZTAKI is based on BOINC, but adds various enhancements to desktop grid computing while the aim of SZDG remains Volunteer Computing. One of the enhancements is support for the hierarchical desktop grid concept as described by Kacsuk et al. [7], which allows a set of projects to be connected to form a directed acyclic graph where work is distributed among the edges of this graph. The hierarchy concept

☆ The EDGeS (Enabling Desktop Grids for e-Science) project receives community funding from the European Commission within Research Infrastructures initiative of FP7 (grant agreement Number 211727).

* Corresponding author. Tel.: +36 13297864; fax: +36 13297864.
E-mail addresses: zfarkas@sztaki.hu (Z. Farkas), kacsuk@sztaki.hu (P. Kacsuk).

0167-739X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2010.12.013

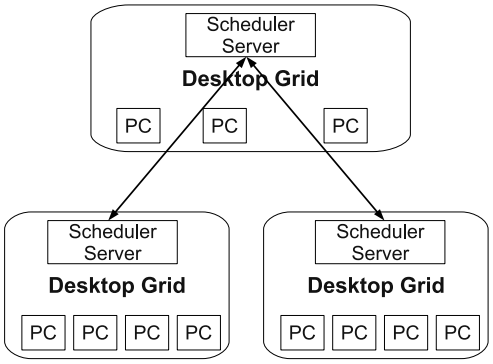

Fig. 1. Hierarchical system example.

is solved with the help of a modified BOINC client application, the Hierarchy Client.

The Hierarchy Client is always running beside any child project, and its only task is to connect to the parent desktop grid, report itself as a powerful client consisting of a given number of processors, and inject fetched workunits into the local desktop grid's database. Generally, a project acting as a parent does not have to be aware of the hierarchy, it only sees the child desktop grid as one powerful client. Additional details of this solution are described in our papers [8,9]. Marosi et al. [9] show how to implement automatic application deployment in hierarchical desktop grid systems, thus administrators of lower level desktop grids do not have to deal with deploying applications of higher level parent desktop grids. An example of a hierarchical system can be seen in Fig. 1.

1.2. Outline of the paper

The main aim of this paper is to examine how child desktop grids can determine their performance, and reflect this in the number of processors reported by the Hierarchy Client. The reported CPU number should result in as low a number of deadline violations and makespans as possible. Minimization of deadline violations is important as this metric reflects how much unnecessary work has been performed. On the other hand, makespan is important as it indicates how "fast" a given algorithm is under certain circumstances.

The paper is organized as follows: Section 2 shows related work in hierarchical structure scheduling, Section 3 introduces algorithms considered, in Section 4 we evaluate scheduling algorithms, finally in Section 5 we conclude our work.

2. Related work in hierarchical structure scheduling

The hierarchical desktop grid concept is a relatively new concept, and as such, there hasn't really been any research that focuses on scheduling questions related to these kinds of systems. Moreover, the desktop grid concept is different from the traditional grid concept: in the latter case schedulers have to send a job with specified requirements to one of the services that most satisfies them, so we can think of them as a system that implements the push model when considering job execution. In the case of desktop grids, the situation changes: resources (clients) contact a central service (the desktop grid server) and fetch some work, thus implement the pull model. As a consequence, the scheduling question changes: how much work should a client fetch for processing, how does the central server concept of BOINC influence scheduling?

Anderson et al. [10] described the task server component of BOINC, and proved using measurements that a single machine can distribute as much as 8.8 million tasks a day, which is much more than the 100 000 jobs the EGEE infrastructure processes a day [11]. So we can say that the single central server concept doesn't introduce a bottleneck.

Regarding client side scheduling, Anderson and McLeod [12] describe BOINC's local scheduling policies, Kondo et al. [13] present some algorithms and compare them with the help of simulation using different properties. Domingues et al. [14] focus on scheduling techniques that improve the turnaround time of desktop grid applications. For this they use a 39-days trace of computer availability of 32 machines in two classrooms, and compare the results with the ideal execution time. Also Domingues et al. [14] have created a tool called DGSchedSim [15] that can be used to evaluate different scheduling algorithms on desktop grids using an existing trace. In paper [16] the authors present a scheduler for desktop grids that is based on stochastic modeling of client availability. Besides this, in his Ph.D. thesis Kondo [17] introduces the cluster equivalence ratio M/N of desktop grids, where this ratio shows how many (M) dedicated cluster machines can be used to represent the performance of a desktop grid consisting of N machines. This ratio depends on application characteristics.

There has been a notable amount of work regarding service grid scheduling. Fibich et al. [18] present the Grid Scheduling Problem: a set of heterogeneous resources, different jobs, constraints and an objective. They also present a model for the scheduling problem. Spooner et al. [19] present the TITAN scheduling architecture that uses performance prediction (performance analysis and characterization environment—PACE [20]), and focus on local scheduling. Weng et al. [21] show a cost-based online scheduling algorithm with a scheduling framework, and analyze the performance of the presented algorithm theoretically, compared to the optimal offline algorithm. The two variants are compared using simulation, too. The work presented by Chapman et al. [22] uses a formal framework based on Kalman filter theory to predict the CPU utilization in clusters. Using the presented predictions the author measured a precision of 15%–20%. Xhafa and Abraham [23] overview computational models for the grid scheduling problem, with service grids in focus.

3. Scheduling algorithms

In this section we overview some possible scheduling algorithms for hierarchical desktop grid systems. We have already described the algorithms in our paper [24] in detail.

An important property of these algorithms is that they are local, that is each child desktop grid runs an instance of one of the scheduling algorithms. The task of the scheduling algorithm is not to send workunits to attached clients, but to determine a number of CPU cores reflecting the performance of the given desktop grid reported by the Hierarchy Client. As the child desktop grid connects to its parent, it will represent itself as powerful client consisting of so many cores, so it will process at most so many workunits originating from its parent in parallel. The reported CPU core number is denoted with $\mathcal{N}$.

Within the scheduling algorithm we assume that each client has one CPU core.

The algorithms are grouped based to the property they operate on: client properties, workunit processing properties, and the local desktop grid's status.

We use the following notations: DG for a desktop grid entity, *DG.nperf* for $\mathcal{N}$, *DG.dgset* for the set of child desktop grids of DG, *DG.CLset* for the client set of a desktop grid, *DG.ldl* for the deadline of the last workunit fetched by DG from its parent, Cl for a client entity, *Cl.lconn* for a client's last connection time, and *Cl.dg* for a client's desktop grid.