Dynamic Self-Scheduling for Parallel Applications with Task Dependencies

Aline Nascimento, Cristina Boeres and Vinod Rebello

e-mail: {depaula,boeres,vinod}@ic.uff.br

Universidade Federal Fluminense
RJ – Brazil

Talk Outline

- Introduction
- Problem Definition
- Related Work
- The EasyGrid Application Management System
- Dynamic Scheduling
- Performance Analysis
- Conclusions

Introduction

- Grid computing are emerging as the platforms of choice to meet the requirements of *e-Science* applications at low cost
- Growth in popularity means that a larger number of applications will compete for limited resources
- Efficient utilisation of the grid infrastructure will be essential to achieve good performance
- It is hard to develop efficient grid management systems
  - Grids are typically dynamic and shared environments, composed of diverse heterogeneous computational resources

Introduction

- Much research is being invested in the development of specialized middleware responsible for
  - Discovering, accessing and harnessing the available and limited resources
- Grid management systems should provide users with a transparent, efficient and robust program execution
- Extracting high performance from grid environments is not trivial
  - Especially for non expert users
- A promising approach is to design autonomic applications

Introduction

- Autonomic applications (self-managing)
  - self-configuring, self-healing, self protecting and self-optimising
- This work bestows MPI applications with a self-optimising property through *self-scheduling*
  - Self-optimising applications are capable of predicting suboptimal behaviours and make adjustments to improve their execution
- This work proposes a distributed dynamic scheduling infrastructure that deals efficiently with precedence constraints
  - There are no implementations of dynamic scheduling heuristic designed specifically for tightly coupled applications in the context of grids

Objectives

- To highlight the scheduling features through the execution of tightly coupled applications
- To show the importance of a novel pro-active and collaborative scheduling approach
- Addresses scalability
  - Minimises scheduling overheads
- To show the viability of the proposed scheduling strategy in the context of an Application Management System
- To quantify the quality of the results

Problem Definition

- Scheduling parallel tasks on a set of resources considering inter-process communication in order to minimize the application *makespan* is known to be NP-Complete

- The *EasyGrid AMS* adopts an *hybrid scheduling* approach
  - Combine the advantages of both static and dynamic schedulers
  - Static Schedulers
    - Estimates assumed *a priori* may be quite different at runtime
    - More sophisticated heuristics can be employed at compile time
  - Dynamic Schedulers
    - Access to accurate runtime system information
    - Decisions need to be made quickly to minimize runtime intrusion

SGC LAB SMART GRID COMPUTING LAB

Problem Definition

- Scheduling heuristics require models that capture relevant characteristics of the application and architecture

- Application Model
  - An application will be represented by a DAG (directed acyclic graph)
  - G = ($V, E, \varepsilon, \omega$)
    - V is the set of vertices that represents tasks
    - E is the set of edges that represents the precedence constraints among tasks
    - ε(v) is the amount of work associated with task v $\in V$
    - ω(u,v) is the amount of data associated with the edge (u,v) $\in E$

Problem Definition

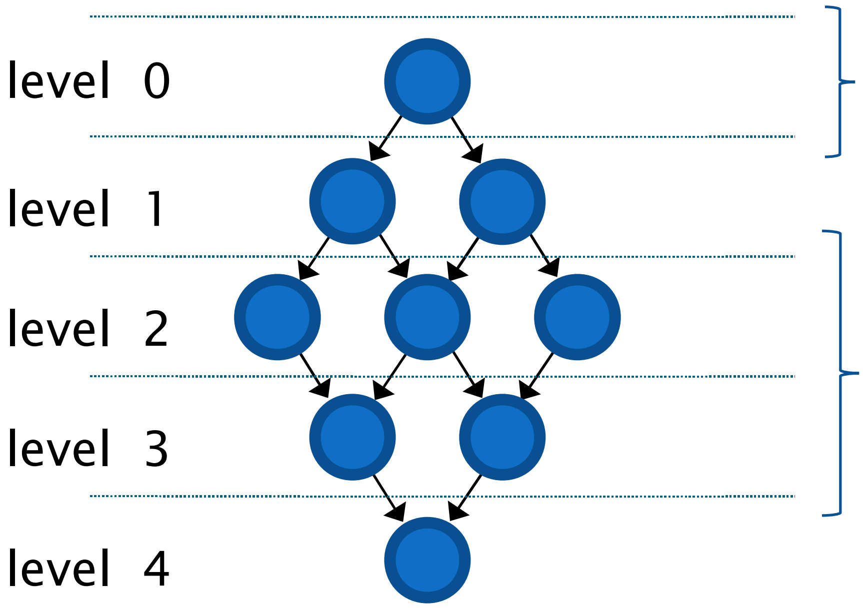
- Architectural model
  - Set R of grid machines
  - While an MPI modelling program will provide the EasyGrid AMS with initial values for the following, during execution the AMS also calculates:
    - computational slowdown index (*csi*) that indicates the maximum computational power of each resource
    - communication delay index (*cdi*) that estimates the latency cost associated with each communication link
- Other concepts
  - estimated execution time $et(v,r_j) = \varepsilon(v) \times csi_j$
  - estimated communication time $comm(u,v) = \omega(u,v) \times cdi(r_j,r_k)$

SGC LAB SMART GRID COMPUTING LAB

Problem Definition

In order to minimise scheduling overhead, dynamic schedulers analyse only a block of the graph

- Other Concepts
  - Topological level: *level*(v)



Diamond Graph

smallest block B_ℓ

$B_\ell = \{v \in V \mid level(v) = \ell\}$

$B_\ell = \{v \in V \mid k \leq level(v) \leq \ell\}$

$B_\ell = \{v \in V \mid level(v) \leq \ell\}$

Related Work

- Most of the work on developing management systems for grid environments has focused on *bag-of-tasks* applications
- Regarding scheduling heuristics
  - Minimize makespan: Max-min, Min-Min, Sufferage, etc
  - Grid economy: considers the total amount that a user wishes to pay to have their application executed in a determined interval
- Some grid management systems are able to execute parallel applications with task dependencies
  - Scheduling heuristics are quite simple: re-schedule only ready tasks ignoring the effect on subsequent tasks

SGC LAB SMART GRID COMPUTING LAB

Related Work

- Overlooked by the majority of grid management systems, there are dynamic scheduling heuristics designed specifically for parallel applications with task dependencies
- Hybrid Scheduling Heuristics
  - PS: Minimum Partial Completion Time Static Priority
  - CS: Minimum Completion Time Static Priority
  - CD: Minimum Completion Time Dynamic Priority
- The dynamic phase of these algorithms uses priorities previously calculated by the static scheduler
  - However they do not consider the initial predefined allocation

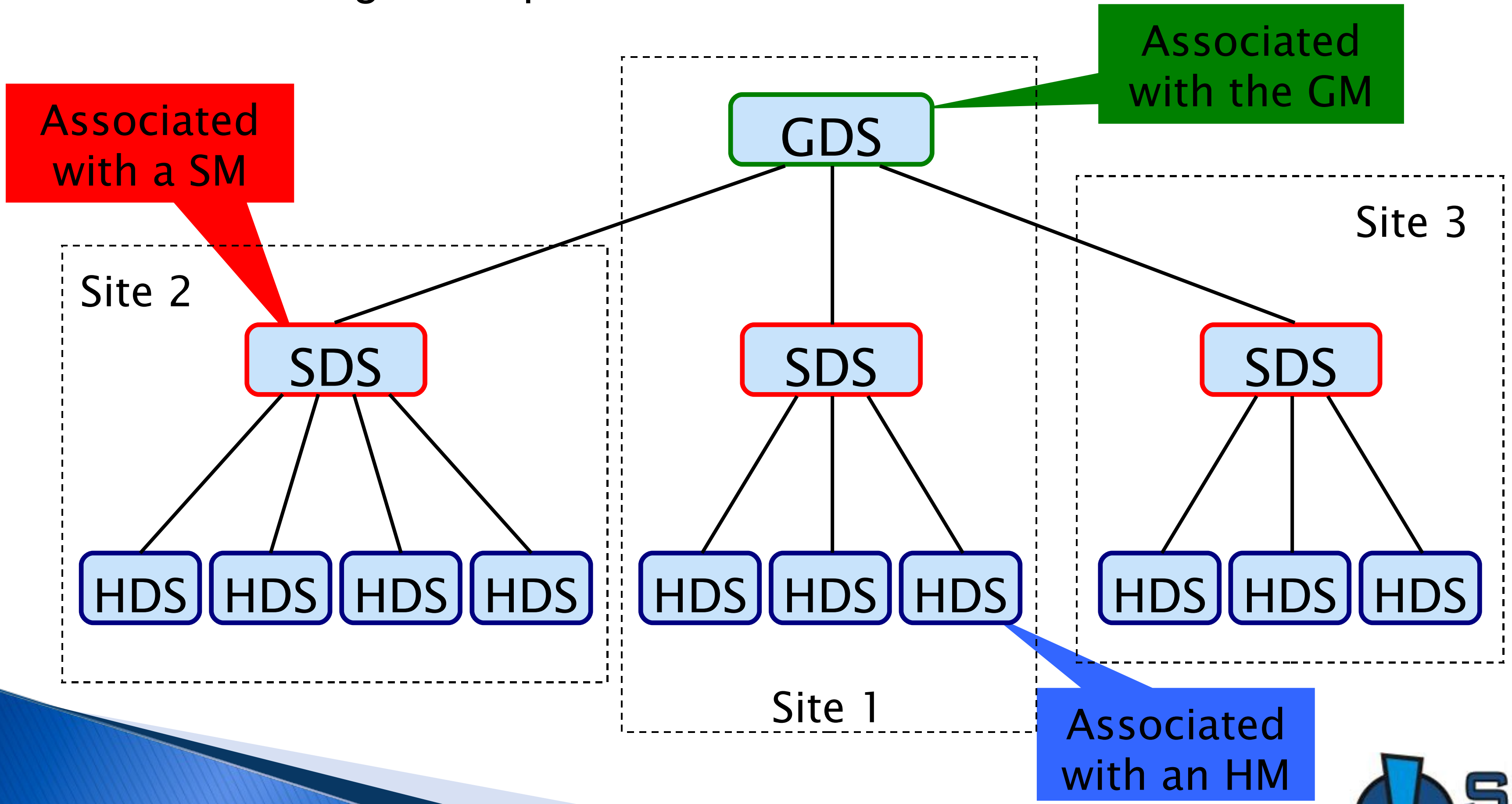
EasyGrid AMS

- It is a Grid Application Management System for MPI implementations with dynamic process creation
- It is automatically embedded into the MPI parallel application, offering better portability
- Requires only Globus and the MPI library to be installed
- Employs a distributed hierarchy of management process
  - Each management process has specific functions: *process management*, *application monitoring*, *dynamic scheduling* and *fault tolerance*
- Each MPI application has its own three level hierarchical management system
  - Decentralised among applications, addressing scalability

SGC LAB SMART GRID COMPUTING LAB

Dynamic Scheduling Structure

- The dynamic schedulers are associated with each of the EasyGrid AMS management processes



Dynamic Scheduling

- The hierarchical scheduling infrastructure has two essential features: flexibility and scalability
- Different policies may be used in different layers of the hierarchy and even within the same layer
- The dynamic schedulers collectively
  - Estimate the remaining execution time on each resource
  - Verify if the allocation needs to be adjusted
  - If necessary, activate the rescheduling mechanism
- A rescheduling mechanism characterises a scheduling event

SGC LAB SMART GRID COMPUTING LAB

HDS – Host Dynamic Scheduler

- HDS determines both the order and the instant that an application process should be created on the host
- Possible scheduling polices to determine process sequence include
  - The order specified by the static scheduler
  - Dataflow: selects any ready task
- A second policy is necessary to indicate when the selected process may execute
  - Influenced by local usage restrictions

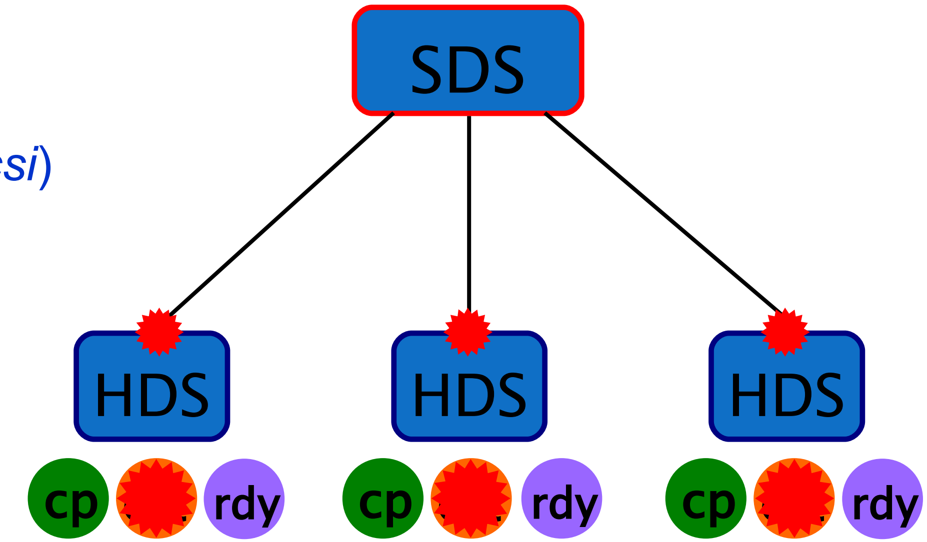
HDS – Host Dynamic Scheduler

- When an application process terminates on a resource, the monitor makes available to the HDS the process's
  - wall clock time and CPU execution time

the percentage CPU utilization
+
computational slowdown index (*csi*)

Resource's current computational power (*cp*)

Estimated remaining time (*ert*)



SDS – Site Dynamic Scheduler

- Treats ready and pending tasks in distinct scheduling events
  - It is possible to assign different priorities and frequencies to the scheduling events by considering specific needs of each group
- Ready Tasks
  - The need of re-scheduling is analyzed in accordance with a predefined interval of time
  - All the ready tasks are considered to re-scheduling
- Pending Tasks
  - The need of re-scheduling is verified in relation to the graph topology (level)
  - Only pending tasks with topological level $\leq \ell$ are evaluated

SDS – Site Dynamic Scheduler

- After receiving cp ert and rdy for all site resources, the SDS:
  1. Creates an auxiliary scheduling structure, $aloc_{\ell}$
  2. Estimates the start and finish times of each task in $aloc_{\ell}$ and consequently estimates the current makespan
  3. Identifies, in order to minimize the current makespan:
     - the set of tasks that should be re-scheduled, Task
     - the set of resources that should give tasks, Rask
     - The set of resources that should receive tasks, Rsub

SDS – Site Dynamic Scheduler

Ready Tasks

- The SDS calculates
  - the target site estimated execution time (sert*)
  - the site imbalance index
- Creates Rask, Rsub and Task where
  - Rask is composed by all sites resources with ert > sert*
  - Rsub is composed by all sites resources with ert < sert*
  - Task is composed by all tasks which finish times are greater than sert*

SGC LAB
SMART GRID COMPUTING LAB

SDS – Site Dynamic Scheduler

Pending Tasks

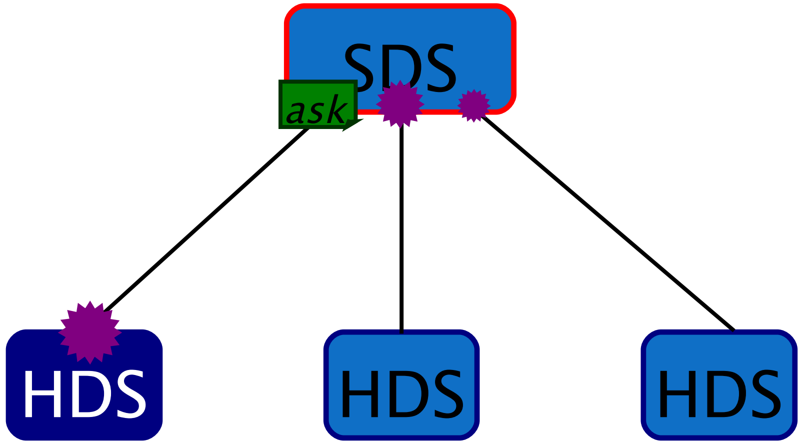
- Based on $aloc_\ell$

  1. Calculates the site estimated makespan (mspan)
  2. Identifies rmax (resource that determines mspan)
  3. Update $aloc_\ell$ re-scheduling tasks in order to minimize the rmax finish time, return to step 1

- Determines Rask, Rsub and Task in step 3 by employing two distinct mechanisms:

  - get critical predecessors tasks from others resources
  - release tasks to under loaded resources in the site

SDS – Site Dynamic Scheduler



- The SDS sends a message
- Each HDSask is responsible for releasing the requested tasks
- The SDS receives the requested tasks and associates each task
- to all HDS ordering the corresponding tasks

SGC LAB SMART GRID COMPUTING LAB

SDS – Site Dynamic Scheduler

- An SDS has no knowledge about the entire system, but it collaborates with the global dynamic scheduling
- When not executing a scheduling event each SDS calculates
  - the average estimated remaining time of the site
  - the sum of computational powers of its resources
- These values are included in the monitoring messages and sent to the Global Dynamic Scheduler (GDS)
- The description of the global dynamic scheduling heuristics are not in the scope of this paper

Performance Analysis

- The experiments were carried out in a real environment
  - It is important to consider actual costs of the overhead caused by the scheduling algorithm and the grid management system
- The grid site was configured with
  - 25 Pentium IV 2.6 GHz processors with 512 Mb RAM, running Linux Fedora Core 2, Globus Toolkit 2.4 and LAM/MPI 7.0.6
- A dedicated environment was established for the experiments
- Controlled background workloads were used to vary the available computational power of individual resources

Performance Analysis

- Three types of task graphs that represent classical parallel applications were used
  - In-trees, Out-trees and Diamond graphs
- Each synthetic application is coded as an MPI program in which is possible to define
  - The amount of work to be performed by each MPI task
  - The amount of communication between MPI tasks
- The main purpose of the tests is to highlight the importance of a dynamic scheduling heuristic that considers not only ready tasks but also their successors

Performance Analysis

- Evaluated Dynamic Scheduling Heuristics
  - CD, CS, PS, GAD and BoT

CD, CS, PS	Same scheduling event for ready and pending tasks Need for scheduling is verified in accordance to the topological level The task block evaluated is composed by all tasks v with level(v) = $\mathcal{L}$
BoT, GAD1	Scheduling event only for ready tasks Need for scheduling is verified in accordance to a predefined interval of time The task block evaluated is composed by all ready tasks v with level(v) $\leq \mathcal{L}$
GAD1	Scheduling event only for pending tasks Need for scheduling is verified in accordance to the topological level The task block evaluated is composed by all pending tasks v with level(v) $\leq \mathcal{L}$

Performance Analysis

- Homogeneous Environment
  - GAD1 and EST achieved the best values
  - BoT was on average 1% worse
  - CD, CS and PS obtained quality values that were from 1% to 7% worse
- Static and Heterogeneous Environment
  - GAD1 and CD achieved the best values
  - EST was on average 45% worse than the best values
  - For diamond graphs BoT was almost 40% worse than GAD1 and CD

Performance Analysis

- Dynamic and Heterogeneous Environment
- Applications with ~256 tasks

t	app	GAD1	CD	CS	PS	BoT	EST
1s	DI	**1.000**	1.052	1.320	1.360	1.548	1.530
	OUT	**1.000**	1.565	1.543	1.574	1.186	1.535
	IN	**1.000**	1.086	1.080	1.113	1.119	1.236
5s	DI	**1.000**	1.065	1.167	1.159	1.556	1.548
	OUT	**1.000**	1.656	1.572	1.561	1.271	1.582
	IN	**1.000**	1.171	1.162	1.172	1.136	1.333

Performance Analysis

- Dynamic and Heterogeneous Environment
- Applications with ~4096 tasks

t	app	GAD1	CD	CS	PS	BoT	EST
1s	DI	**1.000**	**1.000**	1.070	1.077	1.356	1.499
	OUT	1,013	1.581	1.578	1.579	**1.000**	1.445
	IN	**1.000**	1.375	1.385	1.388	**1.000**	1.524
5s	DI	**1.000**	**1.000**	1.040	1.039	1.332	1.431
	OUT	**1.000**	1.582	1.577	1.574	1.002	1.579
	IN	**1.000**	1.387	1.392	1.387	1.019	1.532

Conclusions

- The dynamic scheduling problem for parallel applications with task dependencies has been neglected by most grid management systems
- This work presents a novel dynamic scheduling strategy that deals efficiently with tightly coupled parallel applications
- The scheduling effort is divided in two mechanisms for ready and pending tasks
  - The AMS can address the different needs of each group without increasing scheduling overhead
- The pro-active and collaborative dynamic scheduling strategy contributes to AMS scalability and efficiency