Performance of Scheduling Strategies in Distributed Systems

Eleni D. Karatza
Department of Informatics
Aristotle University of Thessaloniki
Greece

Lisbon, October 19th, 2009
Presentation Structure

• **Introduction**
  – Distributed Systems
  – Workloads
    • Parallel Jobs with Independent Tasks and Groups of Tasks
    • Gangs

• **Scheduling Strategies**

• **Scheduling Real-Time Jobs**

• **Conclusions and Current Directions**
Distributed systems offer considerable computational power, which can be used to solve problems with large computational requirements.

- Several loosely interconnected processors
- Heterogeneous clusters
- Multiple sites
- Grids
Workloads

• Parallel jobs consist of:
  – independent tasks which can execute on any processor and in any order
  – independent groups of tasks
  – tasks which need to frequently communicate with each other – they start essentially at the same time and execute for the same amount of time (gangs).
Scheduling Algorithms

• The most important aspect of a distributed system is the scheduling algorithm.

• The scheduling algorithm is responsible for the allocation of processors to jobs and determines the order in which jobs will be executed on processors.
Periodic (Epoch) Mapping and Scheduling

- Various mapping and scheduling policies for independent tasks and independent groups of tasks have been examined.
- The best methods usually involve a considerable overhead due to their complexity.
- It has been shown that applying the best methods periodically can result in comparable performance and in less overhead.
• References in Periodic (Epoch) Scheduling:


In a distributed real-time system, jobs usually consist of frequently communicating tasks which can be processed in parallel.

An efficient way to schedule dynamic, parallel jobs is **Gang Scheduling**, which is a combination of time and space sharing.

According to this technique, a parallel job is decomposed into tasks that are grouped together into a gang and scheduled and executed simultaneously on different processors.

The number of tasks in a gang must be less than or equal to the number of available processors.
Policies for Scheduling gangs:

Adapted-First-Come-First-Served (AFCFS)
This method attempts to schedule a gang whenever processors assigned to its tasks are available. When there are not enough processors available for a large job whose tasks are waiting in the front of the queues, AFCFS policy schedules smaller jobs whose tasks are behind the tasks of the large job.

Largest-Job-First-Served (LJFS)
With this policy tasks are placed in increasing gang size order in processor queues (tasks that belong to larger gangs are placed at the head of queues). All tasks in queues are searched in order, and the first jobs whose assigned processors are available begin execution.

Relative Performance of AFCFS and LJFS:
It has been shown that in most cases LJFS performs better than AFCFS
Gang Scheduling III

Gang Scheduling in the Presence of Critical Sporadic Jobs.

• Several resource management and scheduling issues arise when using distributed processors for parallel computing when high priority tasks arrive sporadically and have to interrupt gangs in order to satisfy their immediate requirements.

• It has also been demonstrated that parallel applications need a dedicated environment to produce good performance.

• In these cases, the proper scheduling method for gangs has to be employed in order to cope with critical sporadic jobs at the smallest resulting delay of gangs.
Gang Scheduling IV

• **References in Gang Scheduling:**


Imprecise Computations

- **Imprecise Computations** is a technique according to which the execution of a real-time job is allowed to return intermediate (imprecise) results of poorer, but still acceptable quality, when the deadline of the job cannot be met.

- It is assumed that every job is **monotone**, that is the accuracy of its intermediate results is increased as more time is spent to produce them.

- If the execution of a monotone job is fully completed, then the results are precise.

- Typically, a monotone job consists of a **mandatory part**, followed by an **optional part**.
• In order to produce an acceptable result, the mandatory part of the job must be completed.

• The optional part refines the result produced by the mandatory part.

• A job is considered completed (i.e. guaranteed), if it completes at least its mandatory part before its deadline.
Scheduling Real-Time Jobs : Fault Tolerance

• A failure may occur during the execution of a job (due to a transient software fault), with probability $FP$.

• We use application-directed checkpoints: Each job is responsible for checkpointing its own progress periodically (by saving its intermediate results), when the 25%, 50% and 75% of its service time is completed.

• The objective is to guarantee that all parallel jobs that arrive in the system will meet their deadlines, providing high quality results.
Scheduling Real-Time Jobs : Scheduling Policies

• Earliest Deadline First (EDF):
  – the priority value of a job is considered to be its absolute deadline $D$ ($PV = D$).
  – the job with the **earliest deadline** has the highest priority.

• Least Laxity First (LLF):
  – we define as laxity $L$ of a job the difference between the relative deadline of the job and the job's service time ($L = D_R - S$).
  – the priority value of a job is considered to be its laxity $L$ ($PV = L$).
  – the job with the **smallest laxity** has the highest priority.
When a gang $j$ arrives in the system, the scheduler must allocate a processor to each of its constituent tasks.

We allocate to $j$ the processors that can provide for its tasks the earliest estimated start time $EST$. 
Scheduling Real-Time Jobs: Incorporation of Imprecise Computations I

- For the EDF and LLF scheduling algorithms we provide an alternative version (EDF_IC and LLF_IC respectively), which allows imprecise computations.
We employ the notion of notification time $NT$ of a job, which is the difference between the absolute deadline of the job and the job's mandatory part ($NT = D - MP$).
• If the deadline of a job in service is reached and the job has already completed its mandatory part, then the job is not considered lost. On the contrary, it is aborted and we accept the results it has produced until the time of its deadline. In every other case the job is lost.

• If a failure occurs during the execution of a job that has already completed its mandatory part, then there is no need for rollback. On the contrary, the job is aborted and we accept the results produced by its mandatory part (i.e. the ones saved by the job's last checkpoint).
In the case where a failure occurs and the affected job has not completed its mandatory part yet, then it is necessary to roll it back and restart its execution from its last generated checkpoint.

If a failure occurs during the execution of a job that has already reached its notification time, then the job is lost and we abort it, because there is no time to roll it back and restart it from its last checkpoint (it will definitely miss its deadline).
Scheduling Real-Time Jobs - Conclusions

- Incorporating the imprecise computations technique into the scheduling process results in significant performance improvement. Results from our simulation studies have proved the following:
  - The proposed scheduling policies EDF_IC and LLF_IC perform better than EDF and LLF respectively.
  - EDF_IC exhibits promising performance and outperforms all the other three policies.
References


Conclusions - Current Directions

• Proper scheduling is fundamental to performance in distributed systems.

• Our current directions include:
  – Scheduling of DAG-based real-time jobs
  – Energy-efficient scheduling in large distributed systems
Parallel and Distributed Systems Group

- Eleni Karatza  <karatza@csd.auth.gr>
- Zafeirios Papazachos  <zpapazac@csd.auth.gr>
- Stylianos Zikos  <szikos@csd.auth.gr>
- Konstantinos Karaoglanoglou  <kkaraogl@csd.auth.gr>
- Fotis Loukos  <fotisl@csd.auth.gr>
- Georgios Stavrinides  <gstavrin@csd.auth.gr>
- Sofia Dimitriadou  <sofiadim@csd.auth.gr>
Thank you

Email: karatza@csd.auth.gr