Performance Evaluation and Analysis of Large Scale Distributed Systems

Issues, Trends, Problems and Solutions

Eleni D. Karatza

Department of Informatics

Aristotle University of Thessaloniki

Greece

Summer School

COST Action cHiPSet

September 21-23, 2016, Bucharest, Romania

Summary

The scope of this talk is:

- to present state-of-the-art research covering a variety of concepts on performance of large scale distributed systems such as grids and clouds,
- to present resource management issues that must be addressed in order to make grids and clouds viable for HPC,
- to provide future trends and directions in the large scale distributed systems area.

Presentation Structure

- Grid Issues - Cloud Issues
  - Performance Evaluation
  - Resource Management and Scheduling
  - Complex Workloads
- Conclusions and Future Direction

Grid Issues - Cloud Issues I

- Parallel computers used in HPC are not always sufficient to cope with resource intensive scientific and commercial applications.

- **Grids** have emerged as an important infrastructure for serving demanding jobs and evolved to become the **basis of Cloud computing**.

Grid Issues - Cloud Issues II

- Computational and data grids and clouds are **large-scale distributed systems used for serving very large and complex applications**.

- Grids and Clouds performance became more important due to the tremendous increase of users and applications.

Grid Issues - Cloud Issues III

Important issues that must be addressed:

- Efficient scheduling,
- Resource management,
- Load balancing,
- Energy efficiency,
- Reliability,
- Security and Trust,
- Cost,
- Availability,
- Quality of Service.

Grid Issues - Cloud Issues IV

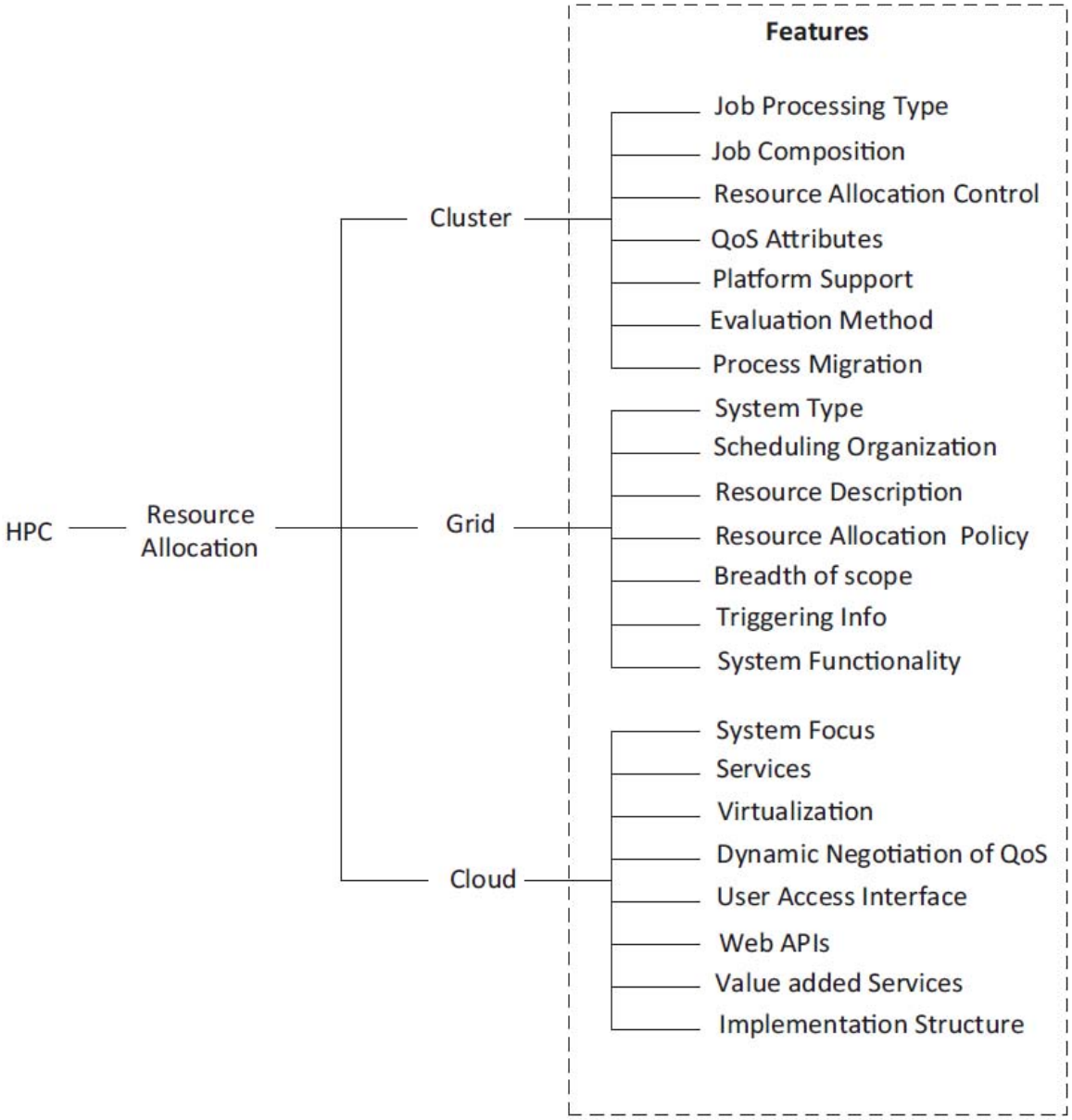


Fig. 1.
HPC systems categories and attributes.

Source: **H. Hussain et als.,** A survey on resource allocation in high performance distributed computing systems, Parallel Computing, Vol. 39, Issue 11, 2013, pp. 709–736.

Grid Computing I

The main idea of Grid Computing:

- To use a large number of distributed high-performance computational resources while minimizing the related operating costs

  in order to solve complex and computationally demanding problems that practically could not be solved on a single resource.

Grid Computing II

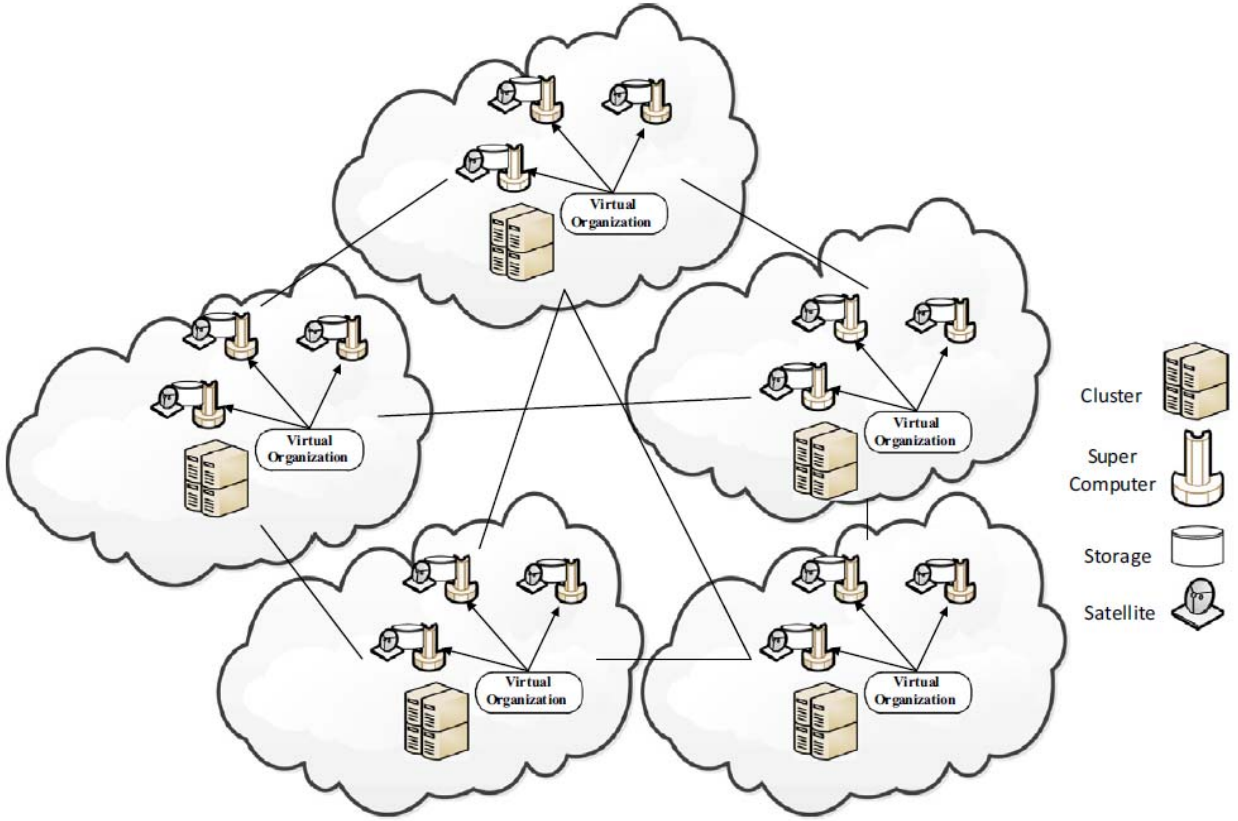


Fig. 2. A model of grid computing system**.**

Source: **H. Hussain et als.,** A survey on resource allocation in high performance distributed computing systems, Parallel Computing, Vol. 39, Issue 11, 2013, pp. 709–736.

Grid Computing III

- In such a dynamic, distributed computing environment, where resource availability varies dramatically, **efficient resource allocation and job scheduling are essential**.

- Grid scheduling manages the selection of the appropriate sites and resources for jobs, the allocation of jobs to specific resources and the monitoring of jobs execution.

Grid Computing IV

- A grid system following a hierarchical architecture is organized at multiple levels:
  - At the grid level, a grid scheduler selects the appropriate sites for jobs
  - At the in-site local level, local schedulers allocate jobs to specific resources.
- Scheduling techniques should perform efficiently across several metrics that represent both:
  - user and system goals.

Grid Computing V

- The usage of **energy** has become a major concern for grid and cloud computing since the price of electricity has increased dramatically.
- **The energy consumption** is a metric aiming at diminishing the energy consumption of computations.
- This metric is always used in multi-criterion optimization problems, otherwise all the jobs would be scheduled sequentially on the most energy efficient machine.

Grid Scheduling I

- **Scheduling** in grid systems has been a major research goal for many years. It has been demonstrated that:
  - Scheduling has a substantial impact on grid systems performance.
  - It can preserve individual application performance so that certain jobs do not suffer from large delays.
  - Simulation models are used to evaluate performance of the policies.

Grid Scheduling II

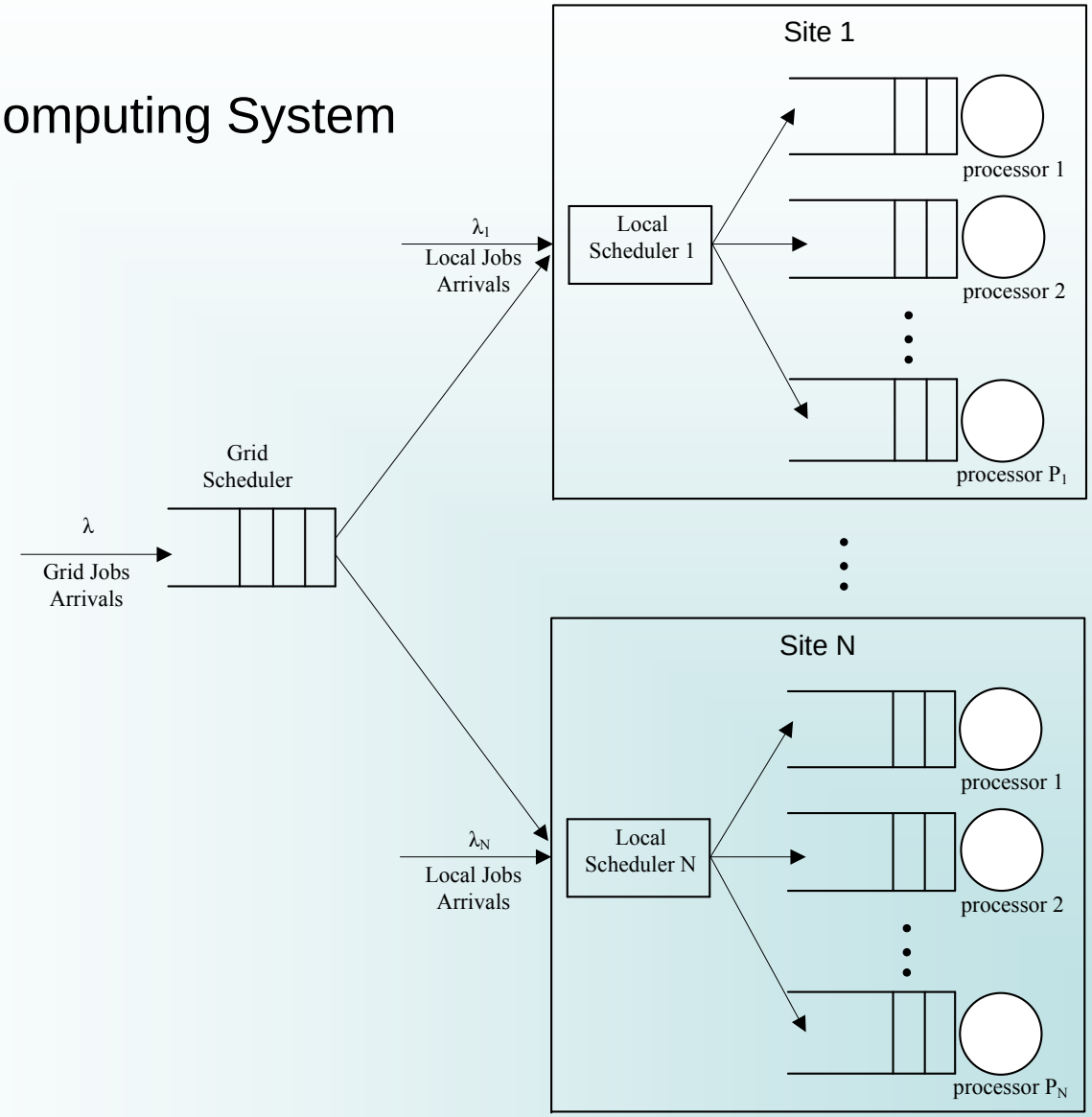
- Grid Scheduling:
  - Grid-Level Scheduling
  - Local-Level Scheduling
- Parallel Job Scheduling
  - BoT Scheduling
  - Gang Scheduling
  - DAG Scheduling
  - Real-Time Scheduling
  - Fault-Tolerant Scheduling

Grid Scheduling III

- Grid systems usually have a hierarchical architecture, consisting of:
-
  - A global, ***Grid Scheduler***, that is in charge of resource discovery and allocation, and job execution management over multiple sites.
  - ***Local Schedulers***, which are responsible for the local scheduling of the jobs allocated to the particular site by the Grid Scheduler.
- At each site, both *grid jobs* and *local jobs* compete for the same resources.

Grid Scheduling IV

Fig. 3. A Grid Computing System



Grid Scheduling V

- Each site consists of a number of processors and a Local Scheduler (LS).
- When a job departs from GS, it arrives at the LS of the selected site.
- The LS submits the job to a processor according to a policy.
- Each processor has its own queue, and a job enters the queue if the processor is busy.

Grid Scheduling VI

Because of the nature of grids, there are important issues that must be addressed:

- Efficient scheduling
- Load balancing.

Heterogeneity problems

- Efficient scheduling was already difficult with homogeneous machines
- In heterogeneous systems, it gets worse – new models are required.

Grid Scheduling VII

Scheduling algorithms usually have to deal with two issues:

- **Resource assignment** which refers to the selection of resources on which each job is assigned.
- **Queue ordering** which refers to the order in which jobs are assigned to resources

Grid Scheduling VIII

Grid scheduling manages:

- the selection of resources for a job,
- the allocation of jobs to resources
- and the monitoring of jobs execution.

In a grid system where two different job types exist, ***grid jobs*** and ***local jobs***, scheduling becomes much more challenging.

Grid Scheduling IX

- Effective load distribution is of great importance at grids as it results to lower response times of jobs and fairness in utilization among the heterogeneous sites.

- All types of jobs (local and grid jobs) must be served as soon as possible, utilizing all the available resources, achieving a **high degree of load balancing among the sites**.

- Load balancing algorithms can be **static** or **dynamic**.

Grid-Level Scheduling Policies I

- **Random:**
  - When there is a job arrival, the Grid Scheduler selects randomly one of the sites and dispatches the grid job to the particular site. In this case the Grid Scheduler queue is not used.

Grid-Level Scheduling Policies II

- **Deferred:**
  - The Grid Scheduler receives feedback from the Local Schedulers regarding their load. This feedback is sent at a specified time interval, which is called ***allocation interval***.
  - When there is a job arrival, the Grid Scheduler stores the job in its queue and scheduling is deferred. The jobs are dispatched from this queue at the end of each allocation interval.
  - For each job, the site with the minimum load is selected. Therefore, load balancing is achieved.
  - The *drawback* of this policy is that grid jobs are delayed in the Grid Scheduler queue, due to the postponement of the scheduling process.

Grid-Level Scheduling Policies III

- **Hybrid GS:**
  - The Hybrid policy is more composite, which combines the Random and Deferred policies (S. Zikos and H. Karatza, 2008).
  - There still exists the concept of Allocation Interval in which the scheduling is deferred until information from sites becomes available.
  - The problem with the Deferred policy is the long delay of jobs that arrive at the beginning of the Allocation Interval in GS. Perhaps it's better to route these jobs to ***a less "good" site*** with zero delay in the GS's queue. This is the case of Random policy.

Grid-Level Scheduling Policies IV

- **Hybrid GS cont.**
  - On the other hand, jobs that arrive in shortly before the end of interval benefit from the best site selection which compensates the delay in queue. This is the Deferred policy part.
  - The question here is when the GS changes policy, from Random to Deferred. A parameter (A_I%) is introduced which shows the percentage of Allocation Interval in which the Random policy is used.
  - If a job arrives beyond the threshold that A_I% defines, then the GS operates according to Deferred policy.

Grid-Level Scheduling Policies V

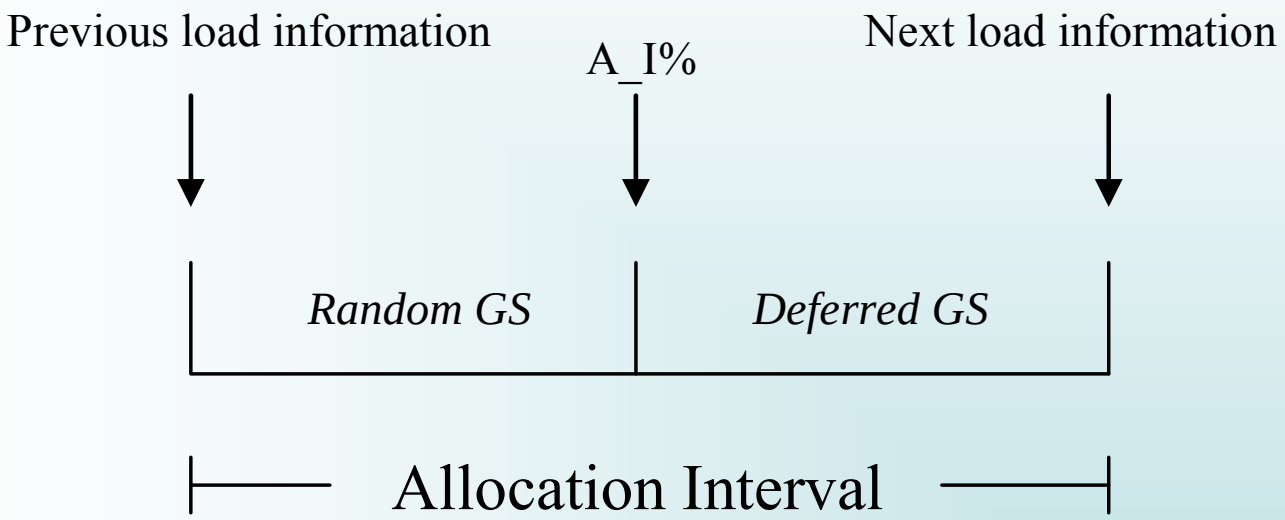


Fig. 4. GS operation when Hybrid_GS policy is used with A_I% = 0.5

Grid-Level Scheduling Policies VI

- **Real-Time:**
  - Like Deferred policy, the Real-Time policy is based on information about each site's load. However, in this case scheduling is not deferred and the Grid Scheduler has updated load information at every job arrival. When a job arrives, the Grid Scheduler allocates it to the least loaded site without delay.
  - This scenario is practically unachievable as the overhead from the continuous feedback traffic would be enormous. It is impossible for the Grid Scheduler to know exactly what is happening to a large number of remote sites. However, this Real-Time policy is often used for comparison purposes.

Grid-Level Scheduling Policies VII

When the optimal percentage of the allocation interval is applied, the Hybrid GS is approaching the Real-Time GS policy in performance.

Grid-Level Scheduling Policies VIII

- **References:**

  - **S. Zikos and H. Karatza**, “Communication Cost Effective Scheduling Policies of Nonclairvoyant Jobs with Load Balancing in a Grid”, The Journal of Systems and Software, Elsevier, Vol. 82, issue 12, 2009, pp. 2103-2116.

  - **S. Zikos and H. Karatza**, “The Impact of Service Demand Variability on Resource Allocation Strategies in a Grid System”, ACM Transactions on Modeling and Computer Simulation (TOMACS), Vol. 20, Issue 4, Article No. 19, October 2010, pp. 19:1-19:29.

  - **S. Zikos and H.D. Karatza**, “Resource Allocation Strategies in a 2-level Hierarchical Grid System”, Proceedings of the 41st Annual Simulation Symposium (ANSS), IEEE Computer Society Press, SCS, April 13-16, 2008, Ottawa, Canada, pp. 157-174.

  - **S. Zikos and H. Karatza**, “Clairvoyant site allocation of jobs with highly variable service demands in a computational grid”, to appear in the Proceedings of the 9th International Workshop on Performance Modeling, Evaluation, and Optimization of Ubiquitous Computing and Networked Systems (PMEO-UCNS'10), in conjunction with IPDPS 2010, April 19-23, 2010, Atlanta, GA.

Local-Level Scheduling Policies I

Resource assignment

- At each site, the Local Scheduler schedules both the local and grid jobs according to a specified scheduling policy.
- Local and grid jobs may consist of a single task or multiple tasks.
- Depending on job characteristics, various types of scheduling policies can be employed in each site.

Local-Level Scheduling Policies II

Resource assignment

- ***Random LS***
  Each LS randomly selects one of the processors to execute a job.

- ***Shortest Queue LS***
  A LS uses information about the number of jobs in each local queue and selects the processor with the least number of jobs waiting in queue. In case there are two empty queues, the idle processor is selected.

- ***2 Random – Shortest Queue LS***
  Two random processors are selected initially, and then the Shortest Queue policy between these processors is applied. In [Mitzenmacher, 2001], it is proven that two choices, instead of one, offer exponential improvement in a job's response time in various models.

Local-Level Scheduling Policies III

Resource assignment

- M. Mitzenmacher, "**The Power of Two Choices in Randomized Load Balancing",** IEEE Transactions on Parallel and Distributed Systems, Volume 12 , Issue 10, (October 2001), pp. 1094 – 1104, 2001.

- **S. Dimitriadou and H.D. Karatza,** "Multi-Site Allocation Policies on a Grid and Local Level", Proceedings of the Fourth International Workshop on Practical Applications of Stochastic Modelling (PASM'09), (Mascots 2009 Workshop), 24 Sept. 2009, Imperial College, London, Elsevier's ENTCS (Electronic Notes in Theoretical Computer Science).

Local-Level Scheduling Policies IV

Resource assignment

- Implementing migrations is a strategy used for dynamic load sharing.
- Migration enables us to dynamically adjust the schedule and rearrange jobs in the queues.
- In heterogeneous multi-cluster systems the need of migration for load balancing purposes is necessary since the load of each cluster might differ in time.
- It is a complex procedure which involves overheads. Therefore, it should be employed with caution in order to be beneficial.

Local-Level Scheduling Policies V

Resource assignment

- **FCFS scheduling policy**

  FCFS ensures certain kind of fairness, does not require in advance information about job execution time, does not require much computational effort, and is easy to implement.

- **Shortest-time-first**

  This policy assumes that a priori knowledge about a task is available in form of service demand. However, a priori information is not often available and only an estimate of job execution time can be obtained.

  This method is not fair to jobs with large service demands. For this reason **modified versions** are used which limit the number of times a large job can be bypassed by smaller jobs.

Parallel Job Scheduling

- A job may consist of independent tasks which can be processed in parallel **(Bag-of-tasks Scheduling).**
- A job may consist of frequently communicating tasks which can be processed in parallel (***Gang Scheduling***).
- A job may be decomposed into a collection of tasks with precedence constraints among them. These tasks may be scheduled on different nodes of the system (***DAG Scheduling***).
- Composite jobs may have end-to-end deadlines (***Real-Time Scheduling***).
- Software failures may occur during the execution of a job (***Fault-Tolerant Scheduling***).

Bag of Tasks Scheduling

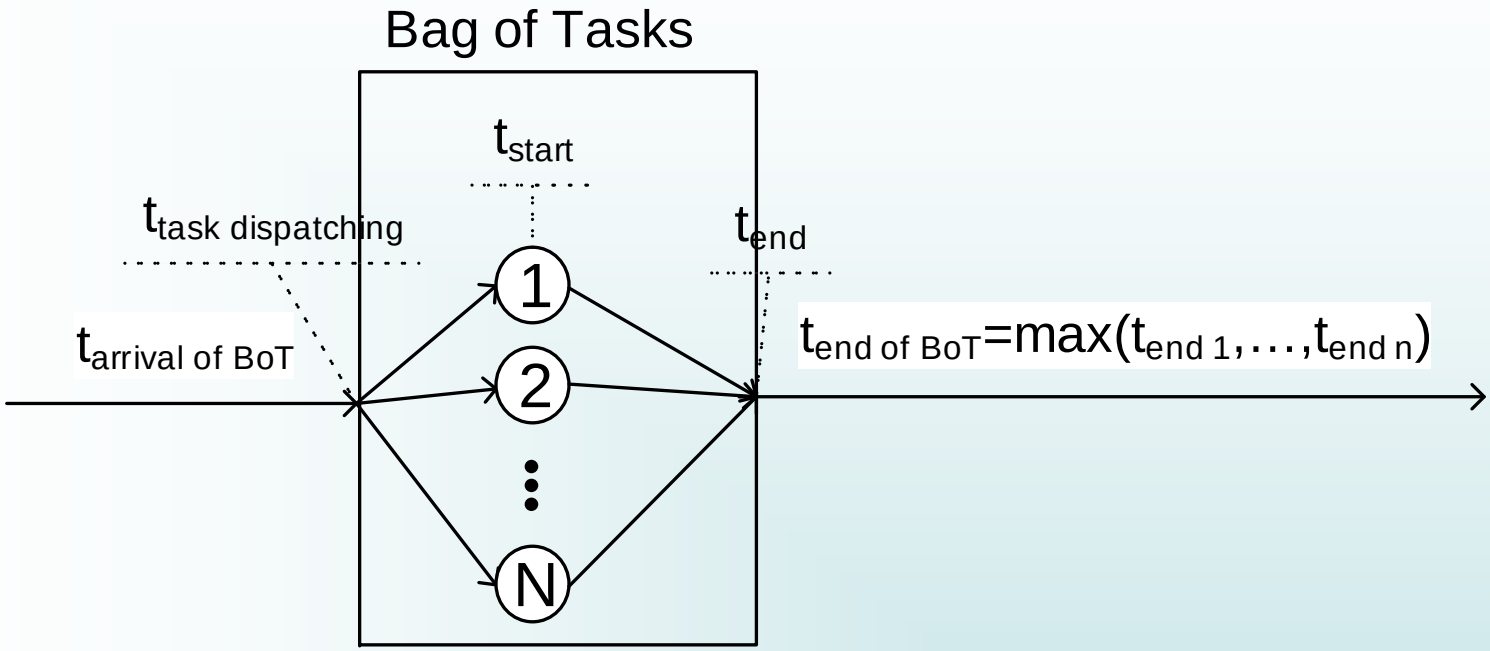


Fig. 5. A BoT

- A BoT is a job which consists of simple independent tasks which arrive to system on the same time.
- Execution of a BoT is completed when all of the tasks which belong to the same job are executed.

Gang Scheduling I

- In distributed systems jobs often consist of frequently communicating tasks which can be processed in parallel.
- An efficient way to schedule this kind of jobs is ***Gang Scheduling***, which is a combination of time and space sharing.
- According to this technique, a job is decomposed into tasks that are grouped together into a gang and scheduled and executed simultaneously on different processors.

Gang Scheduling II

- The overhead of the communication between the tasks of a job is assumed to be included in the execution time of the tasks.
- The number of tasks in a gang must be less or equal to the number of available processors.

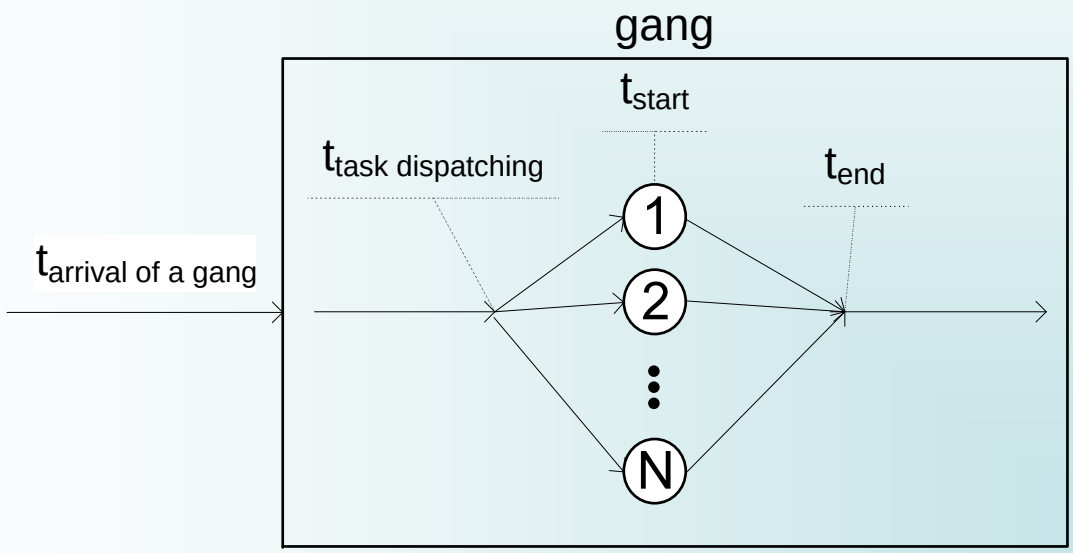


Fig. 6. Model of a gang job with N tasks

Gang Scheduling III

- In Gang Scheduling, the tasks of a job need to start execution simultaneously, because in this way the risk of a task waiting to communicate with another task that is currently not running is avoided.
- Without Gang Scheduling, the synchronization of a job's tasks would require more context switches and thus additional overhead.
- In Gang Scheduling, in order for a job with N tasks to be completed, N processors must execute the tasks concurrently.

Gang Scheduling IV

- A common problem that occurs when using gang scheduling in a Grid, is that there is a number of processors in all sites that remain idle even if there are tasks in their queues.
- This is due to the fact that the execution of a gang does not start unless all of its tasks can start execution simultaneously.
- ***Task Migrations*** is a technique to avoid this delay.
- ***Migration*** is the transfer of a gang task from one queue to the head of another queue.

Gang Scheduling V

- The tasks of a gang can be migrated both **locally** and **through the grid**. (Papazachos and Karatza 2009a, b)
- A local migration is the transfer of a task from one processor queue to another, in the same cluster (site).
- A grid migration is the transfer of a task from one cluster to another. A grid migration causes a higher overhead than a local migration.
- However, the overhead caused by migrations during gang scheduling in a grid can be tolerable and migrations can ultimately lead to a better system performance.

Gang Scheduling VI

In order for the scheduler to allocate a gang that is ready for execution to a set of processors, one of the following policies is usually used:

- **Adapted First Come First Served (AFCFS):**
  - Attempts to schedule a job (gang) whenever processors assigned to its tasks are available.
  - When there are not enough processors available for a large job whose tasks are waiting at the front of the queues, it schedules smaller jobs whose tasks are behind the tasks of the large job.
  - One major problem of this policy is that it tends to favor jobs consisting of a smaller number of tasks.

Gang Scheduling VII

- **Largest Job First Served (LJFS):**
  - Tasks are placed in increasing job size order in processor queues. That is, tasks that belong to larger jobs (gangs) are placed at the head of the queues.
  - All tasks in queues are searched in order and the first jobs whose assigned processors are available begin execution.
  - This method tends to improve the performance of large, highly parallel jobs at the expense of smaller jobs, but in many cases this is desirable (e.g. supercomputer centers often run large, highly parallel jobs that cannot run elsewhere).
  - It has been shown that in most cases LJFS performs better than AFCFS. However, the relative performance of the two scheduling strategies depends on the system and workload models.

Gang Scheduling VIII

- There are often complex parallel jobs which are bags of independent gangs instead of single gangs.
- **A Bag of Gangs (BoG)** completes its execution when every gang which belongs to the job finishes its execution.
- The gangs of a BoG can be executed on any processors in any order.
- The price paid for the increased parallelism is a synchronization delay that occurs when gangs wait for other sibling gangs of the same BoG to finish their execution.
- The performance of scheduling BoGs in a system with heterogeneous clusters is examined in Papazachos and Karatza, 2010. It is shown that LGFS with migrations outperforms the other methods.

Gang Scheduling IX

References

- **Z. Papazachos and H. Karatza**, “Performance Evaluation of Gang Scheduling in a Two-Cluster System with Migrations”, Proceedings of the 8th International Workshop on Performance Modeling, Evaluation, and Optimization of Ubiquitous Computing and Network Systems (PMEO-UCNS 2009) IEEE International Parallel & Distributed Processing Symposium (IPDPS), Rome, Italy, May 25-29, 2009a.

- **Z. Papazachos and H. Karatza**, “Gang Scheduling in a Two-Cluster System with Critical Sporadic Jobs and Migrations”, Proceedings of the 2009 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2009), Istanbul, Turkey, 13-16 July, 2009b, pp. 41-48.

Gang Scheduling X

- **References**

- **S. Dimitriadou and H.D. Karatza**, “Job Scheduling in a Distributed System Using Backfilling with Inaccurate Runtime Computations”, Proceedings of the 4th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS-2010), IEEE Computer Society, Krakow, Poland, Febr. 15-18, 2010.

- **Z. Papazachos and H. Karatza**, “Performance Evaluation of Bag of Gangs Scheduling in a Heterogeneous Distributed System”, Journal of Systems and Software, Elsevier, Vol. 83 (2010), pp. 1346–1354.

- **Z. Papazachos, and H. Karatza**, “Scheduling Bags of Tasks and Gangs in a Distributed System”, The 2015 International Conference on Computer, Information and Telecommunication Systems (CITS 2015), Gijón, Spain, July 15-17, 2015.

DAG Scheduling I

- In grid systems, a different workload model from gangs, is the following:
- A job may be decomposed into a collection of tasks with ***precedence constraints*** among them, so that a task's output may be used as input by other tasks of the job.
- That is, a job is a ***Directed Acyclic Graph (DAG).***
- In order for a task to start execution, all of its predecessor tasks must have been completed.

DAG Scheduling II

- A task with no predecessors is called an ***entry*** task, whereas a task with no successors is called an ***exit*** task.
- The immediate predecessors of a task are called ***parents*** of the particular task.
- The immediate successors of a task are called ***children*** of the particular task.
- Each vertex in a DAG represents a task of the job, whereas a directed edge between two tasks represents a message that must be sent from the parent task to the child task.

DAG Scheduling III

- Assigned to each edge is a communication cost.
- A child task can start execution only if it has received its required ***input data*** from ***all*** of its ***parent tasks***.
- It is assumed that no additional data are required during a task's execution and that the output data of a task are available only after the task's completion.
- If a child task is assigned to the same processor as one of its parent tasks, the communication cost between the two tasks is considered negligible.

DAG Scheduling IV

- The ***level L*** of a task is the length of the longest path from the particular task to an exit task of the graph, taking into account *the execution times of the tasks and the communication costs of the edges on the path*. The level of an exit task is equal to the task's service (i.e. execution) time.

- The ***static level SL*** of a task is the length of the longest path from the particular task to an exit task of the graph, but *without taking into account the communication cost of the edges on the path*.

- The ***critical path length CPL*** of the DAG is the length of the longest path from an entry task to an exit task in the graph. That is, it is equal to the *highest level L of a task in the graph*.

DAG Scheduling V

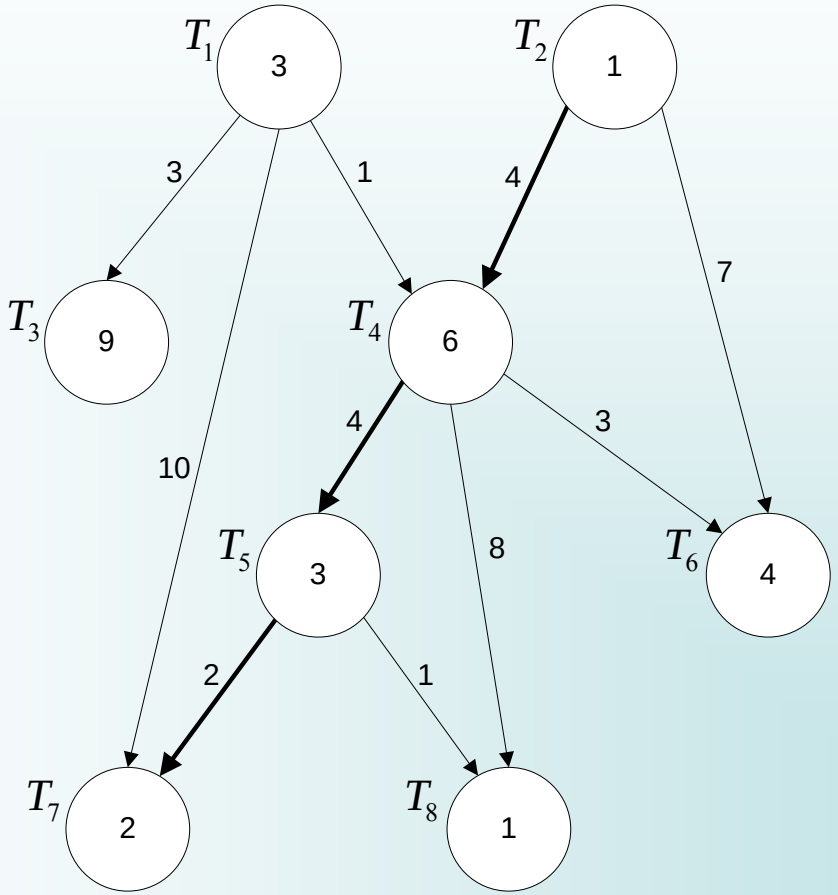


Fig. 7. A Directed Acyclic Graph (DAG)

DAG Scheduling VI

Task	Level L	Static Level SL
T_1	21	14
T_2	22	12
T_3	9	9
T_4	17	11
T_5	7	5
T_6	4	4
T_7	2	2
T_8	1	1

DAG Scheduling VII

- The critical path of the DAG is *CPL*=22. The edges on the critical path of the graph are shown with thick arrows.

- Some of the most commonly used DAG Scheduling policies are:

- **Highest Level First (HLF):**

  – The ready task with the highest level *L* is scheduled first.

DAG Scheduling VIII

- **Dynamic Level Scheduling (DLS):**
  - At each step, this policy selects the ready task T_i - processor P_j pair which gives the *largest value to the expression:*
    - $SL(T_i) - EST(T_i, P_j)$
  - where $SL(T_i)$ is the static level of task T_i and $EST(T_i, P_j)$ is the *earliest estimated start time* of task T_i on processor P_j .
  - The above expression is called the ***dynamic level*** of the task – processor pair (T_i, P_j).

DAG Scheduling IX

- It has been proved that in the case of the static scheduling of a single DAG on a multiprocessor system, the time complexity of HLF is $\boldsymbol{O(n^2)}$, whereas the time complexity of DLS is $\boldsymbol{O(pn^3)}$, where n is the number of tasks in the DAG and p is the number of processors in the system.

- Therefore, DLS causes a higher scheduling overhead than HLF, since at each scheduling step it performs exhaustive pair matching of ready tasks to processors, in order to find the task – processor pair with the highest dynamic level.

- Ultimately, the relative performance of HLF and DLS depends on the system and workload models and on the performance metric that is used. Examples of performance metrics are the total makespan (schedule length) and the processor utilization.

DAG Scheduling X

- **G.L. Stavrinides and H.D. Karatza**, “Scheduling Multiple Task Graphs with End-to-End Deadlines in Distributed Real-Time Systems Utilizing Imprecise Computations”, Journal of Systems and Software, Elsevier, 83 (2010) pp. 1004–1014.

Real-Time Scheduling I

- Grid systems are often used to run **real-time applications**.
- In ***real-time systems*** the correctness of the system does not depend only on the logical results of the computations, but also on the time at which the results are produced.
- Such systems are used for the control of nuclear power plants, financial markets, radar applications and wireless communications.
- The jobs in a real-time system have ***deadlines*** which must be met.
- If a real-time job cannot meet its deadline, then its results will be useless, or even worse, catastrophic for the system and the environment that is under control.

Real-Time Scheduling II

- **Periodic jobs – Aperiodic jobs**

A periodic job J_i is characterized by (P_i, C_i), where P_i is the period of job J_i and C_i is the execution time of J_i. The deadline of the job is D_i, where $\boldsymbol{D_i \leq P_i}$.

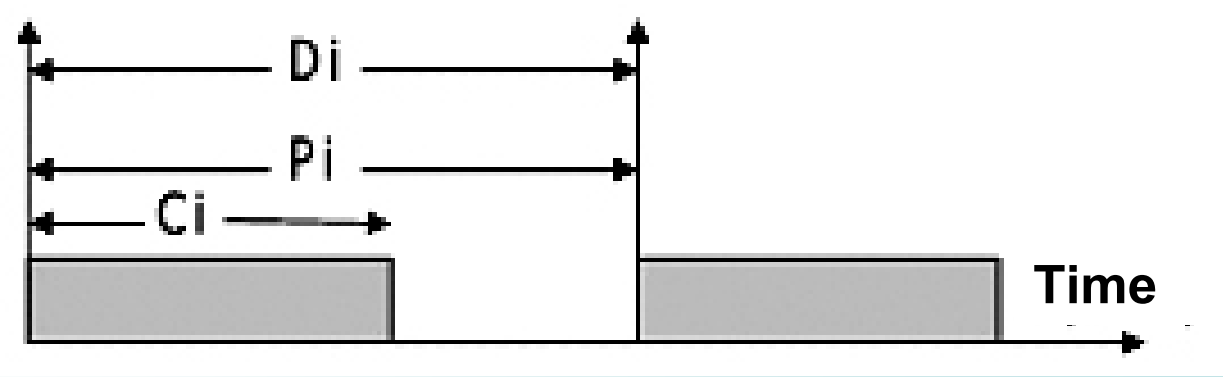


Fig. 8. A periodic job, $D_i = P_i$.

Real-Time Scheduling III

- A real-time system must guarantee that every job will complete its execution before its deadline.

- Some of the most commonly used real-time scheduling algorithms are:

➢ **Earliest Deadline First (EDF):**

  - The job with the earliest deadline is scheduled first.

Real-Time Scheduling IV

- **Least Laxity First (LLF):**
  - The job with the smallest ***laxity*** is scheduled first.
  - The ***laxity LX*** of a job *J* at time *t* is defined as follows:

$$LX_t(J) = D(J) - t - S_t(J)$$

  - where *D(J)* is the absolute deadline of job *J* (measured from time zero) and $S_t(J)$ is the remaining service time of job *J* at time *t*.
  - That is, the laxity of a job *J* at time *t* is the amount of time left between *J*'s completion and *J*'s deadline, assuming that *J* could start execution at the current time instant (time *t*).

Real-Time Scheduling V

- Since the laxity of a job changes over time, it must be recalculated at each scheduling step, causing additional scheduling overhead.

- It has been proved that both EDF and LLF policies are optimal on a single processor with independent, periodic, preemptible jobs. However, neither of these algorithms has been shown to be optimal on multiprocessor systems.

- In most cases of distributed real-time systems, EDF performs better than LLF. However, the relative performance of each strategy ultimately depends on the system and workload models, as well as the employed performance metric (e.g. *job guarantee ratio*, *tardiness* of the jobs etc).

Real-Time Scheduling VI

- In real-time systems it is often more desirable for a job to produce an approximate result by its deadline, than to produce an exact result late.

- ***Imprecise Computations*** can achieve that. It 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.

Real-Time Scheduling VII

- 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 MP***, followed by an ***optional part OP***.
- In order for a job to be completed, it must complete at least its mandatory part before its deadline.

Real-Time Scheduling VIII

The ***notification time NT*** of a job is the difference between the absolute deadline of the job and the job's mandatory part *(NT = D - MP)*.

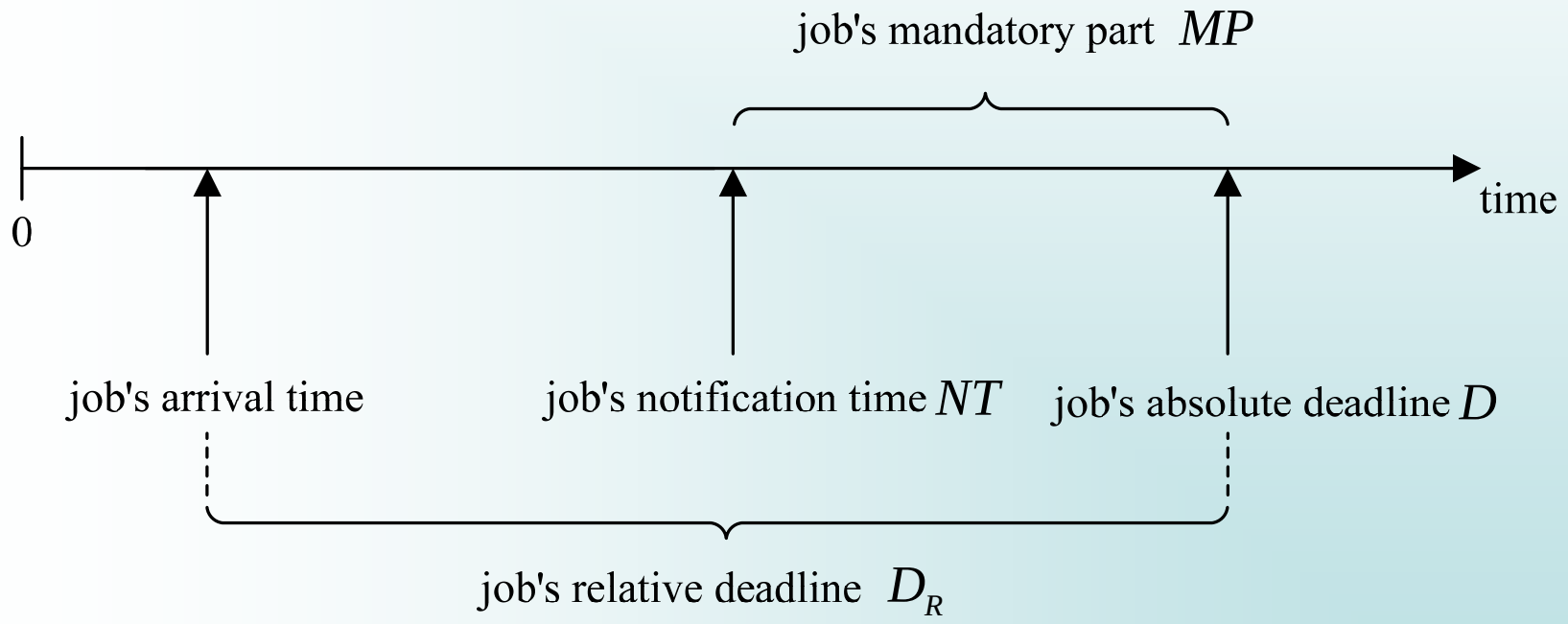


Fig. 9. A job's associated times in the Imprecise Computations case

Real-Time Scheduling IX

- If a job *J* is waiting for service and its notification time is reached, then it can start execution if:
  - its assigned processor is idle or
  - the job in service on *J*'s assigned processor has completed its mandatory part. In this case, the job in service is aborted and job *J* occupies the processor.
- If job *J* cannot start execution, it is considered lost, because it will definitely miss its deadline.

Real-Time Scheduling X

- In the case where a real-time job is a DAG and has an end-to-end deadline, the following scheduling policy can be used:

➢ **Least Space-Time First (LSTF):**

  - The ready task with the smallest ***space-time*** is scheduled first.

  - The ***space-time ST*** of a task *T* (which is a member of a DAG *J*) at time *t* is defined as:

$$ST_t(T) = D(J) - t - L(T)$$

  - where *D(J)* is the absolute deadline of job *J* and *L(T)* is the level of task *T*.

Real-Time Scheduling XI

- In the Imprecise Computations case, the output of a parent task in a DAG may be imprecise.
- Therefore, the child tasks that use as input the result of the particular parent task may have input error.
- Input error may cause an increase in the execution time of the mandatory part of a child task, since more time may be required by the child task to correct the error and produce an acceptable result.
- The quality of a DAG's results ultimately depends on the result precision of the DAG's exit tasks. Therefore, all exit tasks of a graph should be allowed to complete their entire optional part.

Real-Time Scheduling XII

- **G. L. Stavrinides and H. D. Karatza**, “Performance evaluation of gang scheduling in distributed real-time systems with possible software faults”, In Proc. of the 2008 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS’08), Edinburgh, UK, Jun. 2008, pp. 1–7.

- **G. L. Stavrinides and H. D. Karatza**, “Fault-tolerant gang scheduling in distributed real-time systems utilizing imprecise computations”, Simulation: Transactions of the Society for Modeling and Simulation International, Sage Publications, vol. 85, no. 8, pp. 525–536, Aug. 2009.

Fault-Tolerant Scheduling I

- Fault tolerance is an important issue in Grid Computing as the availability of Grid resources can not be guaranteed.
- Real-time systems in particular, need to tolerate possible ***software faults*** that may cause failures during the execution of a job.
- Imprecise computations combined with checkpointing can provide fault-tolerance in distributed real-time systems.
- This is achieved with ***application-directed checkpoints***:
  - each job is responsible for checkpointing its own progress periodically (by saving its intermediate results) at regular intervals during its execution, so that a checkpoint takes place when the job completes its mandatory part.

Fault-Tolerant Scheduling II

Example: Checkpoints occur when the 20%, 40%, 60% and 80% of the job's service time is completed. The mandatory part of the job constitutes the 60% of the job's service time. The third checkpoint takes place when the mandatory part of the job is completed.

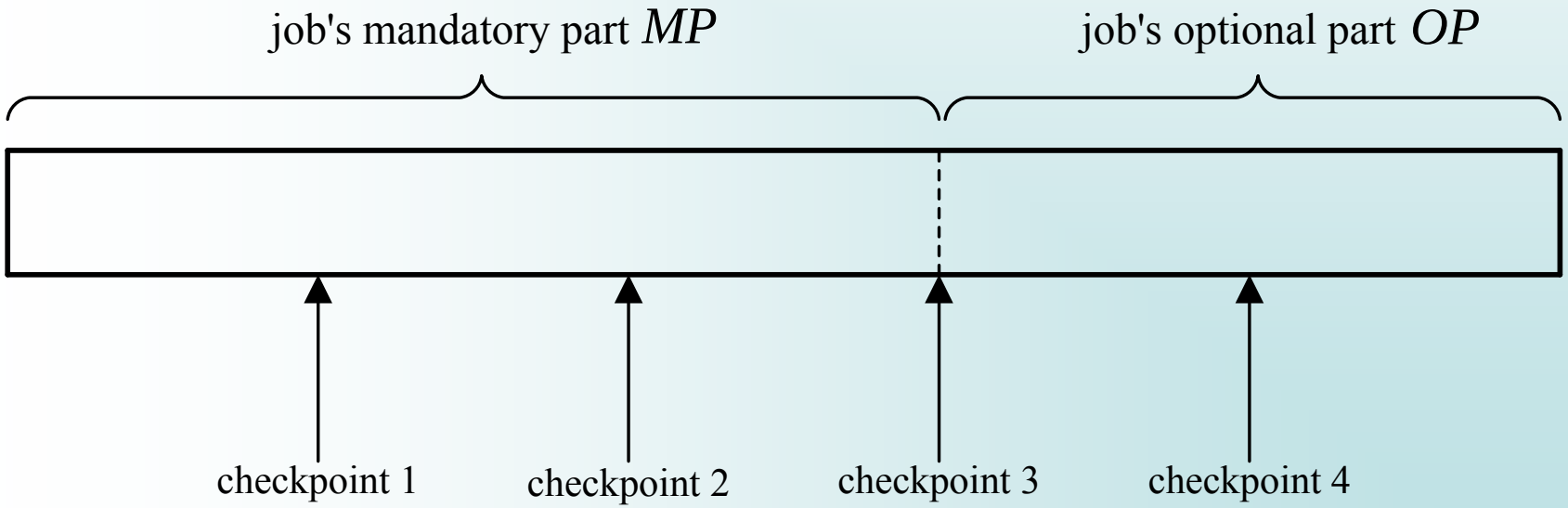


Fig. 10. Checkpoints

Fault-Tolerant Scheduling III

- When a failure occurs, the interrupted job is rolled back and resumes execution from its last generated checkpoint.

- If the last generated checkpoint of the interrupted job occurred after the completion of the job's mandatory part, then there is no need for rollback. The job is aborted and we accept the imprecise results saved by the job's last checkpoint.

Fault-Tolerant Scheduling IV

For DAG applications, effective scheduling methods must include fault tolerant mechanisms to preserve the execution in the case of processor failures.

- **G. L. Stavrinides and H. D. Karatza**, “Performance evaluation of gang scheduling in distributed real-time systems with possible software faults”, In Proc. of the 2008 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS’08), Edinburgh, UK, Jun. 2008, pp. 1–7.

- **G. L. Stavrinides and H. D. Karatza**, “Fault-tolerant gang scheduling in distributed real-time systems utilizing imprecise computations”, Simulation: Transactions of the Society for Modeling and Simulation International, Sage Publications, vol. 85, no. 8, pp. 525–536, Aug. 2009.

Cloud Computing I

Cloud computing evolves from grid computing

Cloud computing is the clear architecture of choice for the bulk of information technology needs of the 21st century

Source: Alexander Pasik, IEEE Roundup,the editors blog 2012.

Cloud Computing II

- Cloud computing provides users the ability **to lease** computational resources from its virtually infinite pool for commercial, business, and scientific applications.

- If cloud computing is going to be used for HPC, sophisticated methods must be considered for both parallel job scheduling and VM scalability.

- Furthermore, high-speed, scalable, reliable networking is required for transferring data within the cloud and between the cloud and external clients.

Cloud Computing III

- Clouds were mostly used for simple **sequential applications**. However, recent evolutions enables the HPC community to run **parallel applications** in the Cloud.

- Good resource management policies can provide great improvements on different metrics:

  - maximum utilization of the resources,
  - faster execution times, and
  - better user's satisfaction (QoS guarantees).

Cloud Computing IV

- Users can have access to a large number of computational resources at a fraction of the cost of maintaining a supercomputer center.

- A user can receive a service from the cloud without ever knowing **which** machines rendered the service, **where** it was located, or how many redundant copies of its data there are.

- The term “cloud” appears to have originated with **depiction of the Internet as a cloud hiding many servers and connections**.

Cloud Computing V

Cloud computing is a paradigm in which computing is moving from personal computers to large, centrally managed datacenters – **Questions:**

- How does cloud computing **differ** from Grid computing, and other previous models of distributed systems?
- What **new functionalities** are available to application developers and service providers?
- How do such applications and services leverage pay-as-you-go pricing models and rapid provisioning to meet **elastic demands** ?

Cloud Computing VI

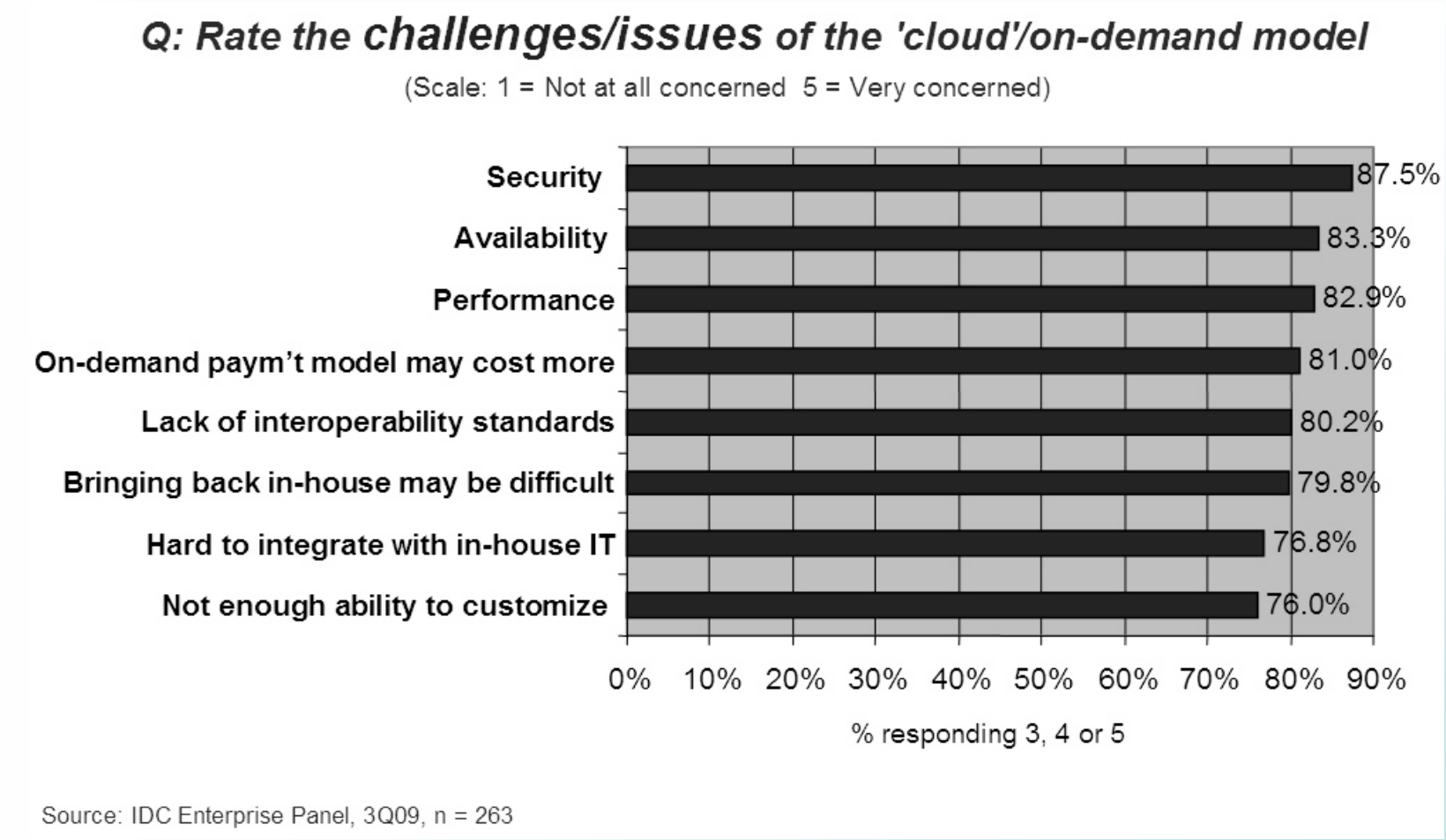


Fig. 11. Cloud Issues (Source: IDC Survey, 3Q09)
http://blogs.idc.com/ie/?p=730

Cloud Computing VII

- The cloud model utilizes the concept of **Virtual Machines** (or VMs) which act as the computational units of the system.
- Depending on the computational needs of the jobs being serviced, new VMs can be **leased** and later **released** dynamically.
- It is important to study, analyze and evaluate both **the performance** and the overall **cost** of different **scheduling algorithms**.

Cloud Computing VIII

- The scheduling algorithms must seek a way to maintain **a good response time to leasing cost ratio.**

- Users requirements for **quality of service** (QoS) and specific system level objectives such as **high utilization**, **cost**, etc. have to be satisfied.

- Furthermore, **data security** and **availability** are critical issues that have to be considered as well.

Cloud Computing IX

Realizing cloud computing benefits requires **new techniques** for:

- managing shared data in the cloud,
- fault-tolerant computation,
- protecting privacy,
- scheduling, and sharing resources among applications,
- communication,
- billing.

Doug Terry, Microsoft, *Chairman, ACM Tech Pack Committee on Cloud Computing, 2011,* http://techpack.acm.org/cloud/

Computing Paradigms I

Today's computing represents the intersection of three broad paradigms for computing infrastructure and use:

(1) **Owner-centric** (traditional) HPC;

- resources are locally owned, with private access

(2) **Grid computing** (resource sharing);

- resources are both locally and externally owned

(3) **Cloud computing** (on-demand resource/ service provisioning).

- resources can be either externally owned (public cloud), or internally owned (private cloud).

Computing Paradigms II

Each paradigm is characterized by a set of **attributes** of the resources making up the infrastructure and of the applications executing on that infrastructure.

- **G. Mateescu, W. Gentzsch, C. Ribbens**

  "Hybrid Computing—Where HPC meets Grid and Cloud Computing", Future Generation Computer Systems 27 (2011) 440–453

Computing Paradigms III

Attribute	HPC	Grid	Cloud
Capacity	fixed	average to high; growth by aggregating independently managed resources	**high; growth by elasticity of commonly managed resources**
Capability	**very high**	average to high	low to average
Virtual Machine Support	rarely	sometimes	**always**
Resource sharing	limited	**high**	limited
Resource heterogeneity	low	**average to high**	low to average
Built-in Workload Management	**yes**	**yes**	no
Distribute Workload Across Resources from Multiple Admin Domains	not applicable	**yes**	no
Interoperability	not applicable	**average**	low
Security	**high**	average	low to average

Source: G. Mateescu et al. / Future Generation Comp. Systems 27 (2011) 440–453

Fig. 12. Comparison of Attributes

Computing Paradigms IV

- Each of the three major computing paradigms has its strengths and weaknesses.
- The motivation for **hybrid computing** is to combine all three paradigms so that strengths are maintained or even enhanced, and weaknesses are reduced.

Cloud Performance – Simulation

Performance Evaluation -Simulation

- The performance evaluation of grids and clouds is often possible only **by simulation** rather than by **analytical techniques**, due to the complexity of the systems.

- Simulation can provide important insights into the efficiency and tradeoffs of scheduling in large-scale heterogeneous distributed systems, such as grids and clouds.

- Synthetic workloads – Traces from real systems.

Cloud Performance - Scheduling

Scheduling manages:

- the **selection** of resources for a job,
- the **allocation** of jobs to resources and
- the **monitoring** of jobs execution.

Cloud Performance – Energy Consumption

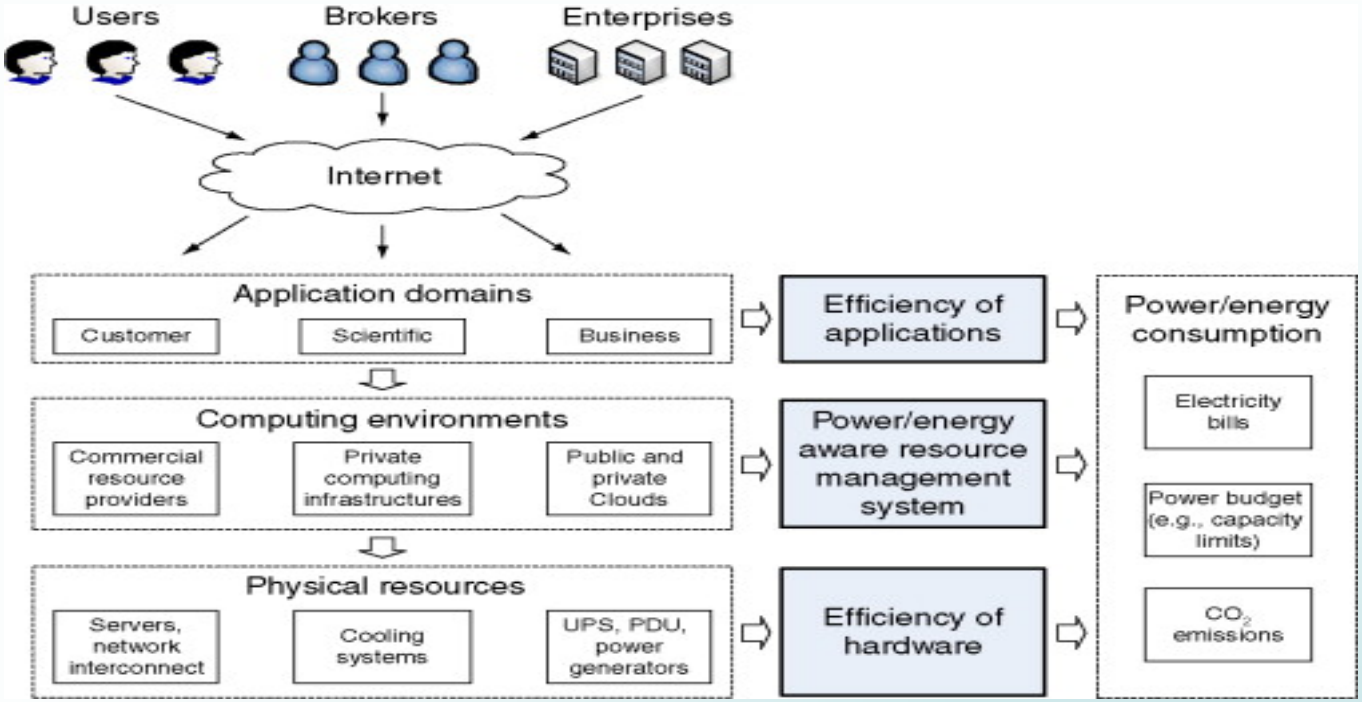


Fig. 13. Energy consumption at different levels in computing systems.

Source: Anton Beloglazov et als., "Chapter 3 - A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems", Advances in Computers, Vol. 82, 2011, 47 – 111.

Cloud Performance – Energy Conservation

Common practices to conserve energy include:

- **Shutting down idle servers**

  The idle power consumption of a server is about 50-60% of its peak power. Turning off the idle servers – versus the recovery time interval required.

- **Use of dynamic voltage scaling (DVS)**

  The voltage is increased or decreased, depending on the system load.

- **Virtualization**

  Many jobs often need only a fraction of the available computational resources. These jobs can be run within a virtual machine (VM) leading to significant increases in overall energy efficiency.

Cloud Performance – Energy Efficiency

Processor characteristics

- Processors are characterized by:
  - Performance capabilities
  - Power characteristics

- Classes of processors:
  - High performance (HP) processors
  - Energy efficient (EE) but slower processors.

Cloud Performance – Environment

Data Centers – Green Cloud

- Data centers hosting Cloud applications consume huge amounts of electrical energy, contributing to **high operational costs** and **carbon footprints to the environment**.

- Therefore, we need **Green Cloud computing** solutions that can not only minimize operational costs but also reduce the environmental impact.

Anton Beloglazova, et als. “Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing”, Future Generation Computer Systems, Vol. 28, Issue 5, May, 2012, pp. 755-768.

Cloud Scheduling – The Simulation Model

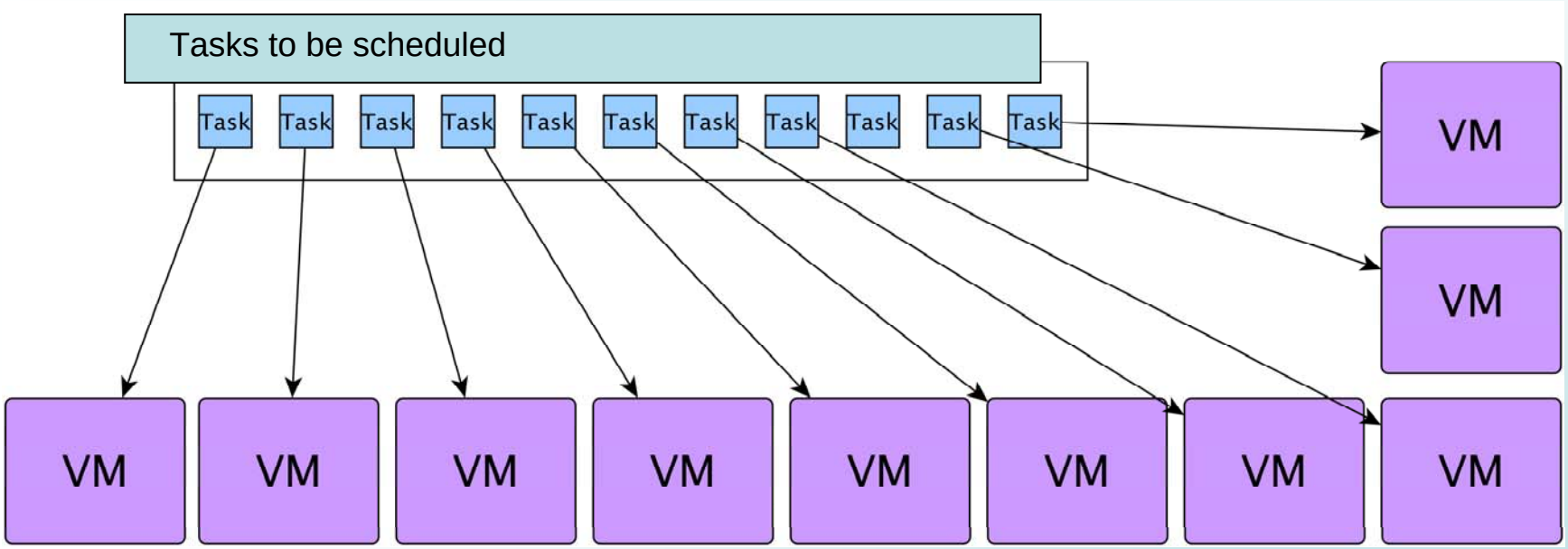


Fig. 14. A task assignment model

Cloud Scheduling – Dispatching

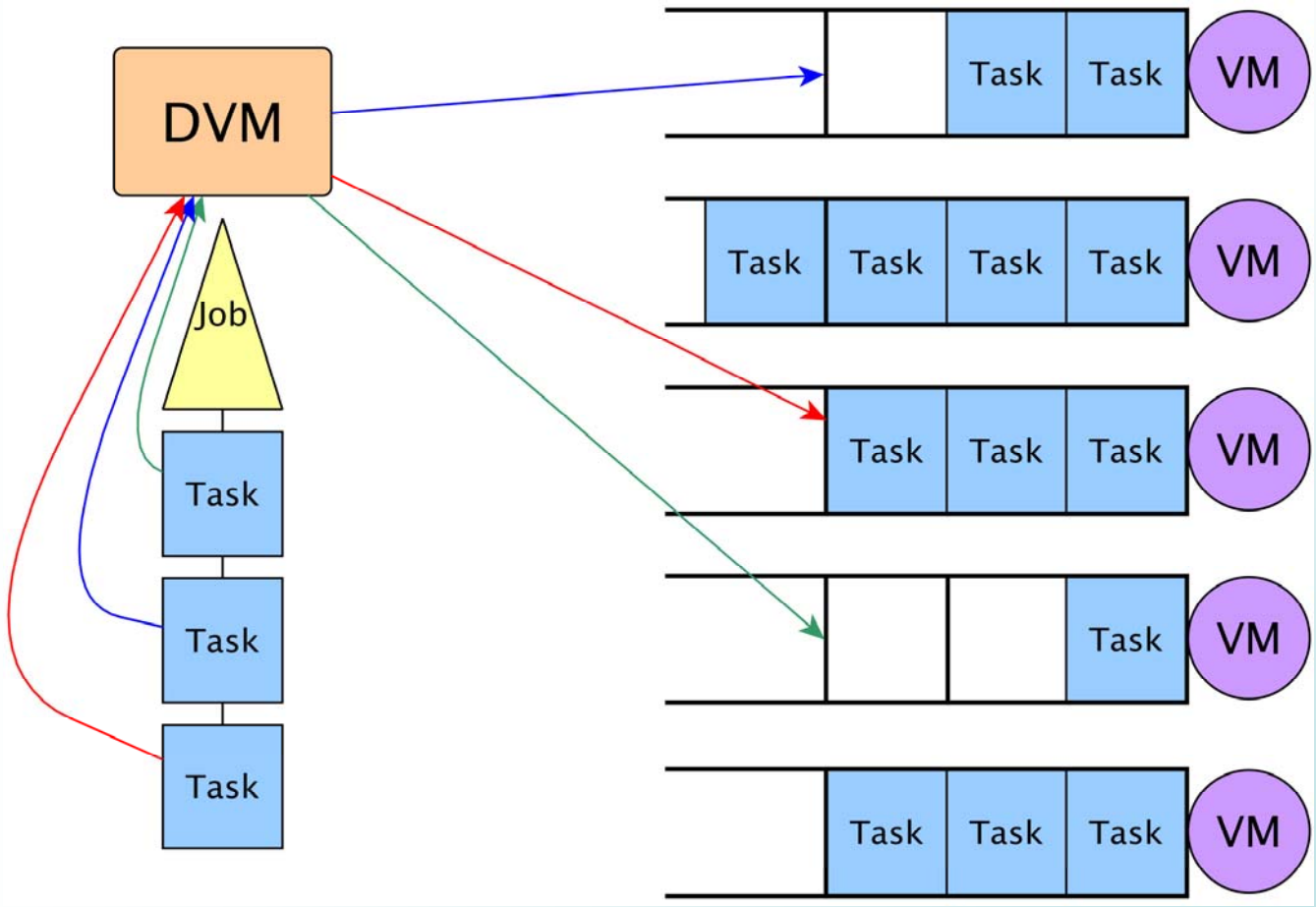


Fig. 15. Job tasks dispatching

Cloud Scheduling – Performance / Cost

- The use of the *Cloud* is "**cost- associative**":
  One pays only for the **computing time** which is equivalent to the **total lease time of virtual machines**.

- ***Cost to performance efficiency* view**.

- *Total lease time* (TL) of virtual machines while the system is in operation:

$$LT = \sum_{i=1}^{P_{tot}} T_{lease(i)}$$

Cloud Scheduling – Migrations I

I. Moschakis and H.D. Karatza, “Performance and Cost evaluation of Gang Scheduling in a Cloud Computing System with Job Migrations and Starvation Handling”, Proceedings of ISCC 2011, June 28-July 1, 2011, Corfu, Greece, pp. 418-423.

Migration and Starvation Handling systems are incorporated to deal with job fragmentation.

Cloud Scheduling – Migrations II

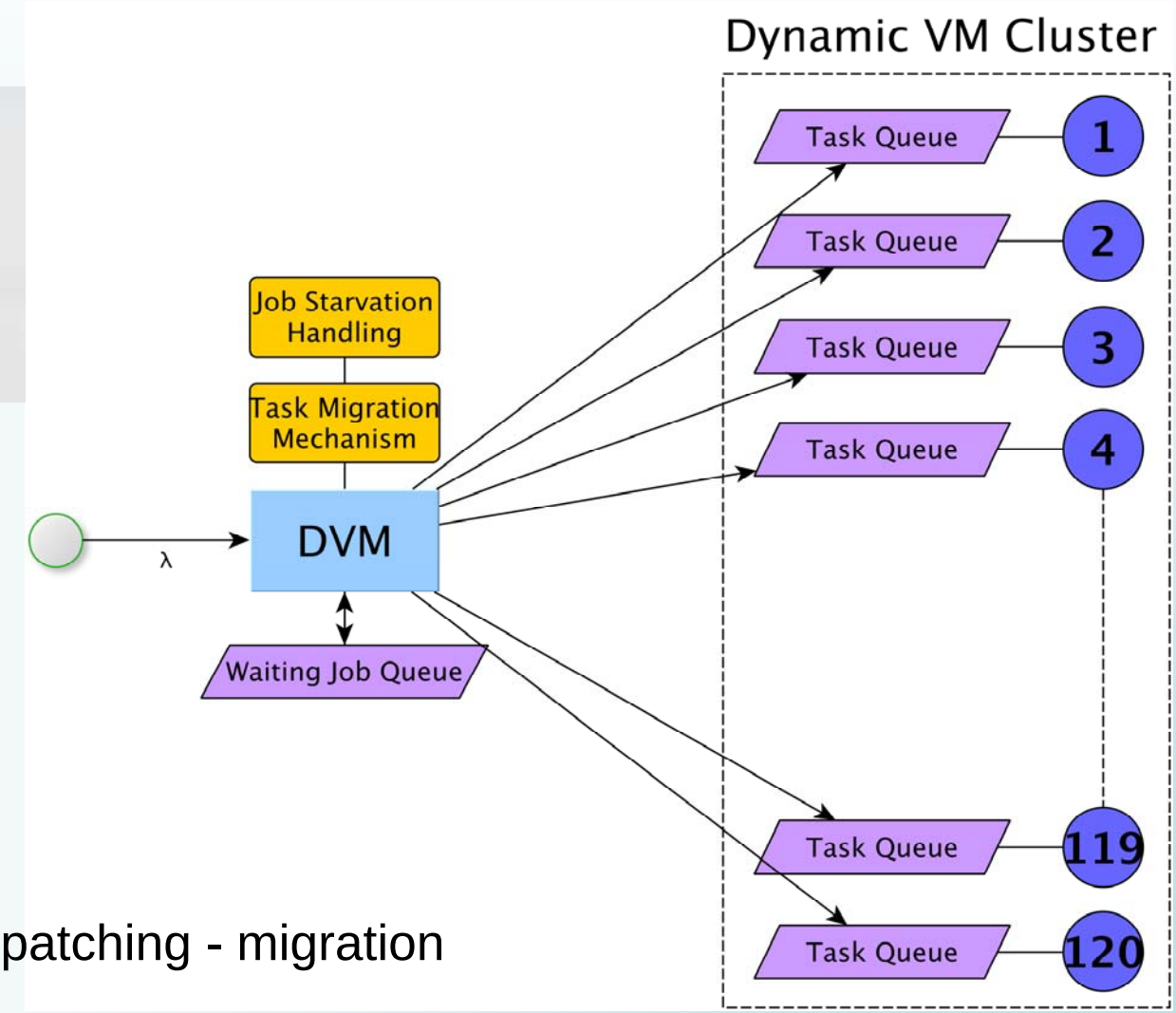


Fig. 16. Tasks dispatching - migration

Cloud Scheduling – Real-Time Workflows

G.L. Stavrinides, H.D. Karatza, “A cost-effective and QoS-aware approach to scheduling real-time workflow applications in PaaS and SaaS clouds”, In Proceedings of the 3rd International Conference on Future Internet of Things and Cloud (FiCloud'15), Rome, Italy, Aug. 2015, IEEE, pp. 231-239.

- Scheduling heuristic for **real-time workflow applications** in a heterogeneous PaaS (or SaaS) cloud.

- **Objectives:**

  (a) to guarantee that all applications will meet their deadline, providing high quality results, and

  (b) to minimize the execution time of each workflow application and thus the cost charged to the user.

Cloud Scheduling – Software Failures

G.L. Stavrinides, H.D. Karatza, “Scheduling real-time parallel applications in SaaS clouds in the presence of transient software failures”, in Proceedings of the 2016 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'16), Montreal, Canada, Jul. 2016.

- **Application-directed checkpointing** and **Approximate Computations**
- **Objectives:**
  (a) provide resilience against temporary software failures,
  (b) guarantee that all applications will meet their deadline,
  (c) provide application results of high quality,
  (d) minimize the monetary cost charged to the end-users.

Cloud Scheduling – Interlinked Clouds I

- **Ioannis A. Moschakis and Helen D. Karatza**, “Multi-criteria scheduling of Bag-of-Tasks applications on heterogeneous interlinked Clouds with Simulated Annealing, Journal of Systems and Software, Elsevier, Vol. 101, March 2015.

- While the use of the **meta-heuristics** does impose a performance overhead due to their complexity in comparison to simpler heuristics,

  the experimental analysis shows that only a relatively small number of steps is required in order to achieve an optimized schedule.

Cloud Scheduling – Interlinked Clouds II

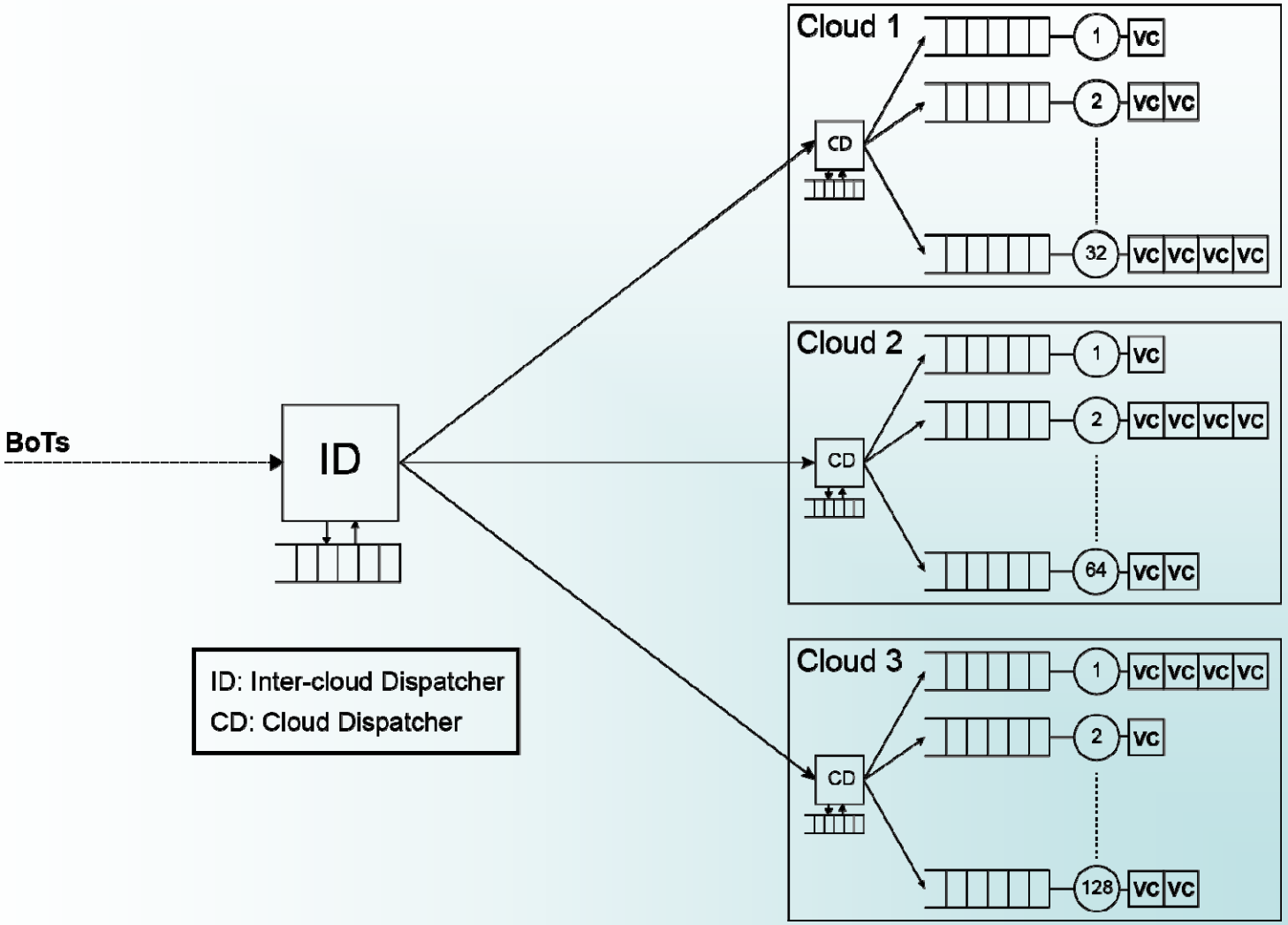


Fig. 17. Interlinked Clouds

From Cloud to Sky Computing I

Sky with Clouds !

Grid Computing:
Aggregation of distributed
heterogeneous resources

Sky Computing:
Aggregation of distributed
heterogeneous Clouds.

Source: Keathey et als., IEEE Internet Computing

Fig. 18. Sky Computing

K. Keahey, M.Tsugawa, A.Matsunaga and J.Fortes, Sky Computing, IEEE Internet Computing, Vol.13, no. 5, 2009, pp. 43-51.

From Cloud to Sky Computing II

- In the past, site owners couldn’t trust a remote resource because they had no control over its configuration.
- Now that **clouds let users control remote resources**, however, this concern is no longer an issue.
- Combining the ability **to trust remote sites with a trusted networking environment**, a virtual site can now exist over distributed resources.

From Cloud to Sky Computing III

- In order to have **different Clouds compatible together**, standards are being developed and also users develop software compatible with multiple Cloud platforms.

R. Buyya, R. Ranjan, and R. Calheiros. InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. LNCS, Vol. 6081 pp. 13–31, Springer Berlin / Heidelberg, 2010.

Jungle Computing

- **Jungle computing** is a form of high performance computing that distributes computational work across cluster, grid and cloud computing.

Jason Maassen, et al, “Towards jungle computing with Ibis/Constellation”, in Proceedings of the 2011 workshop on Dynamic distributed data-intensive applications, programming abstractions, and systems, ACM New York.

Frank Seinstra et al, “Jungle Computing: Distributed Supercomputing Beyond Clusters, Grids, and Clouds”, in Grids, Clouds and Virtualization, Computer Communications and Networks", Springer-Verlag London Limited, 2011.

Conclusions and Future Directions I

- Advances in **processing**, **communication** and **systems/middleware** technologies had as a result:

  -- new paradigms and platforms for computing.

- The **Cloud computing paradigm** promises:

  -- on-demand scalability, reliability, and cost-effective high-performance.

Conclusions and Future Directions II

- Our perception of computing is changing constantly (**Mobile Cloud Computing**).
- The rise of Cloud computing presents a new opportunity for the evolution of computing.
- **Maybe**, in few years computers will be nothing more than thin-clients, and **all our processing will be done on the Clouds**.

Conclusions and Future Directions III

- **Cloud computing** offers **great opportunities for enterprises**.
- **Simulation modeling** is a valuable cost effective tool to efficiently examine **the costs and risks** associated with **moving enterprise applications to the Cloud**.
- By using simulation, organizations **can avoid risks** and they **can estimate in advance** the possible benefits of moving some or all of their applications to the Cloud.

Conclusions and Future Directions IV

- However, **multiple issues have to be addressed** before Clouds become viable for large scale processing like HPC.

- **Security** and **availability** will need the improvement of existing technologies, or the introduction of new ones, in order to achieve scalability that spans a very large number of nodes.

Thank you !

karatza@csd.auth.gr