Analysis of Scheduling Algorithms for Computer Clusters

I. Grudenic and N. Bogunovic
Department of Electronics, Microelectronics, Computer and Intelligent Systems
Faculty of electrical engineering and computing, University of Zagreb
Unska 3, Zagreb, Croatia
Phone: 01-6129-999 int. 548 Fax: 01-6129-653 E-mail: igor.grudenic@fer.hr

Abstract – Most scheduling problems are hard to solve optimally for the reasonable size of the input. These problems are even harder to tackle for distributed computing environments. Since distributed computing is a rapidly evolving trend in data processing, there is increased interest in making this approach efficient.
Organization of heterogeneous resources in a constantly changing environment makes cluster scheduling a challenging task. In this paper we present different scheduling algorithms that are most frequently employed on computer clusters. We recognize different metrics and system architectures, and analyze various approaches to job and resource matching in different environments.

I. INTRODUCTION

Distributed computing is a platform that enables complex computations to complete within acceptable time. The main goal of a distributed system is to provide transparent and scalable resources to the users. In order for distributed system to perform in this manner, proper software must be provided that organizes heterogeneity of machines and user requirements.

Schedulers, as well as resource managers, are responsible for monitoring and orchestrating complex hardware configurations. Optimal scheduling of heterogeneous jobs in heterogeneous environments is known to be NP complete problem [1]. Even more, scheduling in computer cluster environment is highly dynamic since conditions can change at high rates. Algorithms for such an environment must therefore be very fast and should quickly adapt to changes. Additionally, algorithms must scale well since computer clusters are becoming larger and more powerful and there is still shortage of distributed scheduling algorithms. Different clusters can be connected to grids and scheduling between administrative domains is an additional challenge. In the end, peer to peer systems can also be observed as a distributed system with specific tasks.

Design of scheduling algorithms depends on several factors that include target platform, job types and performance metric. In this paper we analyze most utilized algorithms with respect to these factors, while concentrating on computer clusters.

The analysis of algorithms starts with general architecture description in section II. Performance metrics are classified and presented in section III. Section IV contains description of scheduling algorithms with target domains pointed out. Conclusion is given in section V.

II. ARCHITECTURE

Job scheduling on computer clusters is generally preformed in a centralized manner. There is one fronted machine that is dedicated for job submission and scheduling. In some schedulers [2] job submission is enabled on every computation node, but the scheduling process is still performed at the specific cluster node.

In order to schedule pending jobs, scheduler component must have up to date information on available resources and status of the running jobs. This information is provided by the resource manager component that can be integrated with the scheduler or implemented as a standalone module. Resource manager is a component distributed across the cluster computation units that monitors the state of each cluster node and makes collected information available to the subscribed schedulers.

There are alternative distributed scheduling algorithms [3][4] that are aimed toward grid scheduling.

III. SCHEDULER PERFORMANCE METRICS

In order to measure scheduling performance some metrics that define desirable behavior of the computer cluster must be selected. Two general metric types are recognized: user centric metrics and resource centric metrics. User centric metrics favor job execution scenarios that maximize user experience, while the resource centric metrics support schedules that improve system utilization.

Exact determination of the metrics that describe quality of user experience with the computer cluster is not possible because different users can value performance differently. Most common method by which clusters are measured for performance from the users point of view is average weighted response time (AWRT):

$$AWRT = \frac{\sum_{j \in Jobs} w_j \cdot (t_j - r_j)}{\sum_{j \in Jobs} w_j}$$

Where r_j and t_j represent job submission and job completion time for the job j. Priority (weight) of the job j is denoted by w_j. Weight can be set by the user or by the scheduler depending on the user credentials or other information.

Since job weight is not very expressive in terms of user satisfaction, alternative measures such as user defined utility functions [5] are proposed. An example of the utility function provided by the user is presented in Fig. 1.

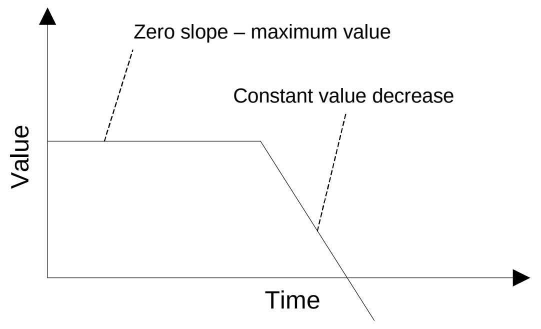

Fig. 1. User utility function

Obviously, if the job is completed instantly the utility function has a maximum value. The utility value which describes user's desire for job to be completed at specific moment decreases as time passes by. There can be zero slopes in utility function that indicate user indifference if the job completes anytime during that period. Indifference periods denote intervals during which user is not available to use the results provided by the completed job. Other user centric metrics are also used in scheduling and are presented with accompanying algorithms.

Resource centric metrics include cluster utilization which can be expressed as percentage of resources used during the analyzed time frame. Cluster throughput is also used to denote scheduling algorithm performance. Makespan is sometimes used as a simplest metric which shows the time needed for the entire job set to complete on the cluster when given scheduler is employed.

Average expansion factor (AXF) that is frequently used when comparing scheduling algorithms is defined as:

$$AXF = AVG\left(\frac{wait\ \ time + run\ \ time}{run\ \ time}\right)$$

where run time and wait time are collected from the set of the scheduled jobs.

It is important to note that scheduler algorithm design is closely related to the targeted metric. Schedule optimization using expressive metrics is usually more computationally expensive then simple metrics optimization.

IV. SCHEDULING ALGORITHMS

Scheduling algorithms are designed to meet several conflicting goals. Some of the goals are high system utilization, maximization of user satisfaction, fairness and scalability. Almost all the algorithms work in a repetitive cycle of the following actions:

```
do forever {
  wait for event;
  obtain updated resource manager information;
  update internal statistics;
  schedule jobs;
}
```

Infinite loop iteration is triggered by any change in the resources state or by the change in the job collection. Change in the job collection can be caused by job insertion, deletion or parameter change. The resource manager information and internal statistics are updated prior to the job scheduling. It is important to note that duration of one loop iteration should be scaled to fit event occurrence rate on the given cluster. Since events on large systems are very frequent it is implied that scheduling algorithm must be very fast and should quickly adapt to changes.

Different scheduling policies and mechanisms can be used to schedule jobs. Following subsections describe common techniques used to schedule jobs. Some of the techniques are used as standalone, while the others are employed together in different manners.

Jobs submitted to the scheduling system can be either sequential or parallel. High occurrence rate of parallel jobs increases complexity in scheduling process which is usually accompanied by decreased cluster utilization. Interactive jobs allow users to interfere and guide job execution.

In order to employ some cluster algorithms, job checkpointing must be supported for the targeted environment. Job checkpointing is a sequence of instructions performed to store the status of the job for restarting. The entire memory space allocated by the process is saved as well as active file descriptors. There are issues with open sockets preservation and active pipes. Checkpoint support [6] is added to jobs in the link phase, and jobs save their own state when they receive a signal from the resource manager.

Traditional straightforward scheduling algorithms are round robin and fairshare algorithm. These are frequently employed in the operating systems designed for one machine. Round robin algorithm favors greedy users that submit large number of short jobs, while fairshare algorithm provides equal access to hardware resources for all the users. Both have their limitations and are not entirely suitable for cluster environments. Sometimes the performance of advanced cluster scheduling algorithms is compared to traditional algorithms because they are easy to implement and measure on all target platforms.

In the following subsections different algorithms, as well as some techniques that enable complexity reduction and performance improvement are presented.

A. Job classes and priorities

Almost all cluster scheduling systems define job classes. These are implemented in the form of queues into which users submit their jobs. Each queue can have limitations regarding the type of jobs it can contain. The constrains can include job parameters such as estimated memory consumption and job duration. Some users or groups of users may be denied to access some queues. Resources in the cluster such as computation nodes and software licenses can also be assigned to queues, effectively limiting the jobs in the queue to employ only the resources available to their respective queue.

Job classes are introduced to enable high level distribution of jobs and resources and to ease the scheduler computational complexity.

Scheduling process scans the queues and finds jobs feasible for execution. Feasible jobs are prioritized using configured policies and history usage. The highest priority jobs are scheduled to available resources. If there are no free resources for the highest priority jobs, reservations are made in order to prevent starvation of important resource consuming jobs.

B. Backfilling algorithm

Backfilling algorithm is used to fill the “holes” that are result of job reservations. When the reservations for high priority jobs are made it is possible for idle time to appear on some of the cluster nodes. Cluster with 2 compute nodes and 4 jobs is presented in Fig. 2. Job priorities are directly related to the job index. Job with smallest index has the highest priority.

On the top of the figure job reservations are made without the use of the backfill algorithm. It can be observed that one processor is idle for two time units before job 3 starts the execution. This idle time can be used to schedule job 4 that fits into the idle time slot.

In order for backfill algorithm to work there must be a significant number of parallel jobs on the cluster. Additionally, each user must provide job runtime estimate.

Conservative backfilling and easy backfilling are two basic types of backfilling algorithm. Conservative backfilling performs reservations for all queued jobs and then allows smaller jobs to move up in the queue provided they do not cause delays in higher priority jobs. Easy backfilling creates reservation only for the highest priority job. The other jobs are then allowed to execute early only if there is no conflict with this job reservation. A comparison of conservative and easy backfilling [7] showed that relative performance depends on the given workload.

An important issue that arises from application of the backfill algorithm is the precision of the user job runtime estimate. It is experimentally shown [8] that smaller error values in job runtime approximation can improve scheduling performance. Larger inaccuracies in the provided job runtime information lead to decrease in

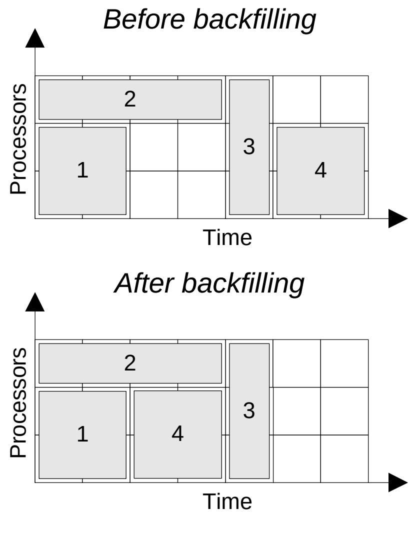

Fig. 2. User utility function

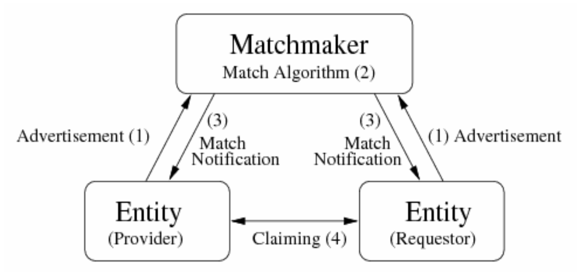

Fig. 3. Matchmaking protocol

overall system utilization.

Advance reservations can be added to the scheduling system in order to increase cluster availability for the specific jobs in the exact time. The impact of advance reservations to both easy and conservative backfill is analyzed in [9]. It is shown that the performance of both algorithms depends on the workload and metric type (system utilization, AXF or AWRT). Easy backfill is measured to achieve better system utilization as advance reservation rate increases with the upper limit of 92%. There was no noticeable utilization change when conservative backfill was employed. The increase in advance reservation rate is followed by higher AXF and AWRT values.

C. Job preemption

In order to immediately execute newly submitted, high importance job, preemption mechanism must be provided. This mechanism allows running jobs to be stopped in order to free the needed resources. Running job preemption can be performed manually or it can be triggered by the cluster policy. Job that is preempted can be only temporarily suspended and its execution can continue if checkpointing is supported by the job and the scheduler. If checkpointing is not available the preempted job is requeued.

Job preemption can be used in combination with the backfill algorithm. When backfilled job is about to exceed the assigned time slot and cause the reservation of the higher priority job to be prolonged, the preemption mechanism is employed. Backfilled job is preempted and its execution can continue or restart in the next available time slot.

D. Matchmaking algorithm

Matchmaking algorithm [10] is designed to work in a highly dynamic settings where resource availability is changed frequently and ownership of resources is decentralized. It is initially targeted for workstation cycle stealing but can be used in dedicated cluster systems. In such an environment instantaneous use of available resource is necessary in order to achieve the best possible performance.

Requestor, provider and matchmaker are three separate entities (Fig. 3.) participating in the matchmaking algorithm. The provider and requestor advertise (1) resources and job requests to the matchmaker in the form of specially designed language. The matchmaker collects advertisements and employs the match algorithm (2) in order to connect requestors with appropriate providers. Match notification (3) is sent back to both the provider and the requestor. Further agreement and potential job execution is their own responsibility. This is done using claiming protocol (4). It is possible for agents involved in the claiming protocol to refuse cooperation due to change in resource or job availability.

The matchmaker operation is not affected by the result of the claiming phase. In the case of claim failure, the requestor or provider can resend advertisement to the matchmaker waiting for the new match to be proposed.

The described matchmaking algorithm deals with only two entities and can be used only for bilateral matching that is shown to be limited in scenarios with several different resources needed for processing of the single job. An example of such a scenario involves a job that must execute on a machine with at least 512MB RAM. For the successful job processing a software license is needed, but the software license is limited to the machines on the specific network domain. Three resources must be matched to each other in this scenario: job with the machine, license with a job, and license with the machine.

Gangmatching algorithm is proposed [11] to address these limitations. Each advertisement contains one or more ports. Ports define interfaces that need to connect with ports of other advertisements in order for matching to succeed. Job advertisement from the above example contains two ports: one for the job-machine match and the other for the job-license match. The goal of the gangmatching algorithm is to create a set of advertisements that connect to each other's ports by obeying to the given constraints. No port should be left unconnected.

There is a naive recursive backtracking search that achieves this goal and is given by the following pseudocode:

```
gangMatch(i_adv, i_port){
 if(i_port>i_adv.nOfPorts) return MATCH;
 if(i_port.bounded==true)
   return gangMatch(i_adv,i_port+1)
 foreach j_adv in availableResources{
   for j_port=1 to j_adv.nOfPorts;{
     if match(i_adv.ports[i_port],
              j_adv.ports[j_port]){
       bound(i_adv.ports[i_port],
             j_adv.ports[j_port]);
       if(gangMatch(j_adv,1)==MATCH)
         gangMatch(i_resource,i_port+1);
       unbound(i_adv.ports[i_port],
               j_adv.ports[j_port]);
     }
   }
 }
}
```

The algorithm is started with the root resource (advertisement) and the index of the first port of that resource. The first port is tested for other ports of the other advertisements until the match is found. Ports are bounded and the new resource is recursively checked for port matches in the same manner. If both ports are bounded successfully and the new resource's ports are matched, the algorithm recursively repeats the step for the next unbounded port. When the last port of the root advertisement is matched, the algorithm is completed and the port matches are transferred to all the involved entities. If port match cannot be found for a specific port, the algorithm backtracks, unbounds the previously matched port and tries to find another solution.

This straightforward algorithm can be further improved in order to speed up the matching. Some improvements include use of index trees in order to find port matches instantly instead of sequentially probing all the possible solutions. Another performance gain is obtained by smart picking of the root advertisement and reordering the ports for the sequential search depending on availability of the resource for the given port.

Gangmatching algorithm is followed by hierarchical claiming phase in which the root entity starts claiming for all the matched resources, and the claiming is continued recursively until the root resource receives reservation confirmation from all its immediate children.

E. Gang algorithm

Previously described algorithms are designed to work on computer clusters with different topologies and architectures. In certain setups diverse algorithms can be used to explore the benefits of the available resources.

Jobs that are submitted for computation can consist of a single thread process or can be made of multiple processes consisted of multiple threads. In the described algorithms reservation of resources for such a job is made through the user request and the job is responsible to use allocated resources as effectively as possible.

Modern processors contain multiple cores and there are computer systems featuring multiple processors connected to the shared memory (SMP systems). These resources can be efficiently employed to minimize communication overheads among different job threads or different job processes.

Gang algorithm [12] is suited for high performance multiprocessor systems where preemption and job resuming doesn't penalize as much as in loosely coupled systems. Each processing elements (processor or processor core) time is divided into discrete timeslots. Duration of the timeslot is determined relative to the cost of the process or thread context switch on the given system. Within one timeslot only one thread can be executed. Threads of the same process are executed simultaneously on different processing elements, but it is a tendency to group them on the "close" processing elements. This group of threads is referred to as "gang". For example, threads of the same process are grouped on the multiple cores of the same processor in order to access shared memory directly and to use processor cache efficiently. The only limitation of this approach lies in the resources availability on the processing element. The sum of all resource requirements of all the threads that timeshare one processing element should not exceed this limitation.

There are many benefits of this approach that are typical for Unix time share systems. Long, resource consuming jobs can be executed without starving other jobs because of timesharing. Interactive jobs can appear to have near real-time performance. High system utilization is achieved for the variety of workloads.

F. Genetic algorithm

Genetic algorithms are traditionally used in different optimization and static scheduling algorithms. Job scheduling on the computer cluster is a dynamic scheduling problem that is highly volatile, especially on large clusters. Events such as job submission, job completion and change in resource availability occur frequently and the scheduling algorithm must perform the entire calculation cycle within two events. It is possible to aggregate events and perform scheduling taking into account the entire new event group, but this leads to inefficiencies in the resulting schedule.

Classic genetic algorithm performs the following steps:

```
Genetic algorithm(){
    initialize population;
    do{
      selection;
      crossover;
      random mutation;
    }until(fitness(best) is good enough);
    return best;
}
```

An initial population is generated [13] using list scheduling heuristics. At first, a small percentage of jobs is assigned randomly to resources, which is followed by mapping of remasining processes to the processors that will finish processing them earliest. This gives reasonably distributed starting population.

The provided fitness function compares theoretical optimal processing time to the estimate of the processing time. Smaller difference between the two yields to higher fitness value. The selection function is a classic roulette wheel function. Chances of an individual to be selected are proportional to the value of the fitness function for the given individual.

The employed crossover algorithm is well known cycle crossover introduced with traveling salesman problem. The mutation randomly swaps task between the processors of chosen individuals. The mutation is completed by performing a re-balancing heuristics to the entire population. The re-balancing heuristics selects processor with the highest load and then the task assigned to this processor is swapped with the shorter task from another processor, but only if a shorter task can be found in five completely random guesses. The five guesses limit bounds on the computation time needed for the mutation to complete.

All the jobs are modeled as single processor jobs with the assigned communication cost $c_{i,j}$ needed to employ task *i* on the processor *j*. It is experimentally deduced that this algorithm performs better than simple heuristics such as round robin, lightest load which schedules task to the processor with lowest current load, earliest first and others. The makespan is shortened by 10%-50% depending on the algorithm compared to. The processor efficiency is shown to be the best for different values of mean communication costs.

This algorithm considers only the subset of unscheduled jobs because the computation of the schedule shouldn't allow processors to become idle. The size of subset varies with the estimation of the occurrence of the first idle processor. In large systems processors become idle frequently which causes the scheduler to map lesser amount of queued jobs to the cluster resources. This improves execution speed of the described genetic algorithm but could result in suboptimal schedules. The impact of reduced target job subsets to the overall cluster performance should be measured.

G. Market based approach

Market based approach [14] to scheduling opposes traditional values in cluster scheduling. A conservative approach in the design of scheduling algorithms puts system utilization as the most import optimization factor. It is not necessary for an algorithm that gets a maximum out of given resources to provide the greatest satisfaction to the users. If optimal utilization causes users to wait 10% longer on the average, then a system with 20% lower utilization but shorter average wait times would be preferred by the users. This leads to increased investment in hardware resources in order to make up for the utilization drop. The general issue is how to obtain stable investment and user satisfaction ratio. This is the type of problem where market based techniques stand out.

If users are made to bid and pay for the resources then the resource price will be a reflection of supply and demand on the simulated market. In times when resources are scarce or when the demand is high, increase in price will force some users to back off either because of the steep pricing or the indifference for the specific timeslot. Market based techniques are shown to be successful on both batch systems as well as in time-sharing systems.

V. CONCLUSION

In this paper we presented an overview of the most frequently used scheduling algorithms. We also addressed the factors that impact their performance and design. Since algorithms are targeted for different workloads, architectures and are employed under different policies, a direct comparison of performance cannot be made. Even more, performance can be measured using different metrics, so in some sections we noted that this can potentially affect the results.

It is generally not feasible to calculate optimal solution for most cluster scheduling problems. On the other hand, there are efficient algorithms that achieve high performance in different metrics under different workload types. The main issue in modern scheduling algorithms is a demand for very fast response which makes complex algorithms not suitable for the job.

Complexity can be introduced in the algorithm if the scheduling procedure can be parallelized. It could also be beneficial to use idle machines in timeslots that could not serve any better purpose. Maybe it would be performancewise to use small percentage of cluster resources even when not idling in order to improve targeted cluster metric.

REFERENCES

[1] D. Bernstein, M. Rodeh and I. Gertner, “On the Complexity of Scheduling Problems for Parallel/Pipelined Machines“, IEEE Transactions on Computers, vol. 38, p. 1308, 1998.

[2] D. Thain, T. Tannenbaum, and Miron Livny, "Distributed Computing in Practice: The Condor Experience" Concurrency and Computation: Practice and Experience, vol. 17, no. 2-4, p. 323, 2005.

[3] M. Moore, “An accurate parallel genetic algorithm to schedule tasks on a cluster”, Journal of Parallel Computing, vol. 30, p. 567, 2004.

[4] W. Zhang1, H. Zhang1, H. He and M. Hu, “Multisite Task Scheduling on Distributed Computing Grid”, Lecture Notes in Computer Science, vol. 3033, p. 57, 2004.

[5] C. B. Lee and A. E. Snavely, “Precise and realistic utility functions for user-centric performance analysis of schedulers”, Proceedings of the 16th international symposium on High performance distributed computing, Monterey, California, USA, p. 107-116, 2007

[6] P. H. Hargrove and J. C. Duell, “Berkeley Lab Checkpoint/Restart (BLCR) for Linux Clusters”, In Proceedings of SciDAC 2006, 2006

[7] S. Srinivasan, R. Kettimuthu, V. Subramani and P. Sadayappan, “Characterization of backfilling strategies for parallel job scheduling”, In the Proceedings of the Parallel Processing Workshops, p.514-519, 2002.

[8] D. Tsafrir, and D.G. Feitelson, “The Dynamics of Backfilling: Solving the Mystery of Why Increased Inaccuracy May Help”, In the Proceedings of IEEE International Symposium on Workload Characterization, p. 131-141, 2006.

[9] B. Li and D. Zhao, “Performance Impact of Advance Reservations from the Grid on Backfill Algorithms”, In the Proceedings of the Sixth International Conference on Grid and Cooperative Computing table of contents, p. 456-461, 2007

[10] R. Raman, M. Livny, and Marvin Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing", In the Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, p. 140, 1998.

[11] R. Raman, M. Livny, and M. Solomon, "Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching", In the Proceedings of the Twelfth IEEE International Symposium on High-Performance Distributed Computing, p. 80-89, 2003

[12] M. A. Jette , “Performance characteristics of gang scheduling in multiprogrammed environments”, In the Proceedings of the 1997 ACM/IEEE conference on Supercomputing, p. 1-12, 1997.

[13] A.J. Page and T.J. Naughton, “Dynamic Task Scheduling using Genetic Algorithms for Heterogeneous Distributed Computing”, In the Proceedings of 19th IEEE International Symposium on Parallel and Distributed Processing, p. 189., 2005

[14] B.N. Chun, “Market-based cluster resource management”, PhD thesis, University of California, Berkley, 2001.