CSIT Conference 2021, Yerevan, Armenia, September 27 - October 1

About the Possibility of Executing Tasks with a Waiting Time Restriction in a Multiprocessor System

Vladimir Sahakyan
Institute for Informatics and Automation Problems of NAS RA
Yerevan, Armenia
email: vladimir.sahakyan@sci.am

Artur Vardanyan
Institute for Informatics and Automation Problems of NAS RA
Yerevan, Armenia
email: artvardanyan@asnet.am

***Abstract*—The most actual problem of organizing computations in a cluster environment is the optimal use of system resources, which is achieved by effective scheduling of the task execution, taking into account the required resources and execution time. This paper considers the priority of acceptance for service, the admissible waiting time. To provide more optimal service, a service model is considered in which service interruptions are performed by creating checkpoints and keeping the interrupted tasks in the queue. As a result, an algorithm was developed for selecting tasks for servicing and determining the moment of the next interruption.**

***Keywords*— Multiprocessor System, Cluster computing, Queueing theory, Multiprocessor Queueing System, Waiting time restriction.**

I. Introduction

The system software of a computing cluster consists of a task control module(schedules and distributes tasks for execution), parallel programming libraries(provide interconnection between the processes of the parallel program) and the operating system in which the processes of the parallel program are executed. The most actual problem of organizing computations in a cluster environment is the optimal use of system resources, which is achieved by effective scheduling of the task execution, taking into account the required resources and execution time [1].

A compute cluster is a multi-user computing environment. Users independently submit their tasks for execution. Tasks are distributed among the nodes and executed. The Job Management System (JMS) is a component of the cluster that performs user task control and scheduling [2]. The job management system basically consists of three components:

- queue manager,
- scheduler,
- resource manager.

The job of the queue manager is to receive job execution requests from users and distribute them to the queue. The queue manager interacts with the scheduler and dispatches tasks to be executed to the resource manager. The queue manager allows users to manage tasks that are in the queue and receive information about the status of execution. In addition, the queue manager takes notes about the use of resources and fixes the execution history of the tasks.

The scheduler determines where and when the task can be completed. To do that, it uses three types of information: job resource requirements, node states, and cluster scheduling and utilization policy. The scheduler receives information about the required resources from the queue manager and information about the state of the nodes from the resource manager [3].

The task Management System Scheduler provides the ability to select standard scheduling algorithms, use custom algorithms, or use an external scheduler. Common scheduling algorithms include FCFS (First Come, First Served), SJF (Shortest Job First), LJF (Longest Job First) [4] and fair-share and backfilling algorithms.

The resource manager also monitors resources, periodically collects information about the state of the nodes, and sends that information to the scheduler as needed.

They include dynamic load balancing, fault tolerance, process migration, priority interruption, and checkpoint creation [5].

Checkpoint creation is the process of saving the state of an active process to the hard drive. The saved state can be used to resume the execution of the process from the savepoint. In [5], an algorithm was proposed that allows interrupting an MPI program implemented using the SPMD model by creating checkpoints, recording the state and then restoring programs for execution.

Provided that processing of big data is required, in particular, using the MapReduce technology [6], partial synchronization of data processing is required. In this case, a processing task arises with a restriction on the waiting time, and for the scheduler, the order of execution of tasks is a priority to take into account the interval in which the task must be serviced. This paper considers the priority of acceptance for service, the admissible waiting time.

To provide more optimal service, consider a service model in which service interruptions are performed by creating checkpoints and keeping the interrupted tasks in the queue.

II. Problem Statement

Suppose that a task stream enters a computing system consisting of m processors ($m \geq 1$). Each task is characterized by three random parameters (ν, β, ω), where ν is the number of computational resources(processors, cores, cluster nodes, etc.,) required to perform the task, β is the maximum time required to complete the task and ω is the possible time that the task can wait before assigning to run, after which it leaves the system without service [4]. Obviously, there are other parameters, but we are not interested in them for solving this problem.

A task can be accepted for service only if it is possible to complete it on time, i.e. the task must leave the system before the moment $\beta + \omega$, when counting from the moment of its acceptance. Otherwise, it receives a denial of service. The aim is to check the possibility of servicing a task if there is a queue of tasks waiting to be serviced in the system.

III. Algorithm

Consider a service discipline in which, at certain points in time, the jobs being serviced interrupt their service and return to the queue for further servicing, i.e. their additional service time is reduced by an amount equal to the time already served. Thus, suppose we have tasks in the queue at some point in time when there was a service interruption.

$$(\nu_1, \beta_1, \omega_1), (\nu_2, \beta_2, \omega_2), ..., (\nu_n, \beta_n, \omega_n)$$

Let us describe an algorithm for selecting service tasks and determining the next interruption time:

Step 1: Let us determine the priorities for servicing tasks. To do this, we sort the queue in ascending order of acceptable wait times. After sorting, we get a queue:

$$\{(\nu_i, \beta_i, \omega_i), \quad i = 1, 2, ..., n\},$$

and moreover

$$0 \leq \omega_1 \leq \omega_2 \leq ... \leq \omega_n.$$

Step 2: We choose for servicing in order of priority from the beginning of the queue as many jobs as possible to serve simultaneously, i.e. we find such k, that

$$\nu_1 + \nu_2 + ... + \nu_k \leq m < \nu_1 + \nu_2 + ... + \nu_{k+1}$$

If $k = n$, then the right part of the inequality is omitted, i.e. all tasks are served. We complete the algorithm, i.e. all tasks will be served.

If that is not the case and after sorting it turns out that $\omega_{k+1} = 0$, then the queue cannot be serviced without losses. We complete the algorithm with the statement that not all jobs can be serviced.

If that is not the case, then we need to add to find out whether it is possible to add jobs with numbers $k+2, k+3, ..., n$ from the queue in order of priority for servicing. If there are such tasks, then add them and enumerate the queue so that k will take a new value. As a result, there will be k tasks selected, and not selected $n - k$.

Step 3: We define the time interval until the next interruption as

$$\tau = min(\beta_1, \beta_2, ..., \beta_k, \omega_{k+1}).$$

Step 4: We calculate the parameters of the queue at the moment of the next interrupt:

$$(\nu_1, \beta_1 - \tau, \omega_1), ..., (\nu_k, \beta_k - \tau, \omega_k),$$
$$(\nu_{k+1}, \beta_{k+1}, \omega_{k+1} - \tau), ..., (\nu_n, \beta_n, \omega_n - \tau).$$

If the moment of interruption was the moment of the end of service, then one of the serviced tasks will leave the system and there will be $n - 1$ of them. If the moment of interruption was not the moment of the end of service, then there will be n tasks in the queue, but for some, the admissible waiting time will become equal to 0, in particular, $\omega_{k+1} = 0$. We return to the initial Step 1.

Obviously, tasks that are being serviced linearly decrease their service time over time, and tasks that are in the queue decrease their admissible waiting time, thereby increasing their priority.

IV. Conclusion

Thus, at the time of interruption, tasks that have shorter admissible waiting times should be served earlier than tasks that can wait longer. If this fails, then the queue cannot be served without loss.

The proposed algorithm can serve as a necessary condition for servicing tasks without interruption, i.e. if a queue cannot be serviced interrupted, then it cannot be serviced without interruption.

The proposed algorithm also conducts a preliminary assessment of the state of the queue in the queuing system(Torque with the Maui scheduler, etc.) and gives recommendations on a possible order of service.

References

[1] Hrachya Astsatryan, Aram Kocharyan, Daniel Hagimont, Arthur Lalayan, "Performance Optimization System for Hadoop and Spark Frameworks", *Cybernetics and Information Technologies*, Sofia, vol. 20, no. 6, pp. 5-17, 2020.

[2] Saeed Iqbal, Rinku Gupta, Yung-Chin Fang, "Job Scheduling in HPC Clusters", Reprinted from *Dell Power Solutions*, 2005.

[3] D. G. Feitelson, L. Rudolph, "Parallel job scheduling: Issues and approaches, IPPS 95 Workshop: Job Scheduling Strategies for Parallel Processing", *Springer-Verlag*, New York, vol. 949, pp. 1-18, 1995.

[4] V. Sahakyan, A. Vardanyan, "The State Probabilities of the System $M|M|m|n$ with the Waiting Time Restriction", *Computer Science and Information Technologies*, Yerevan, Armenia, pp.181-184, 2019.

[5] M. Movsisyan, V. Sahakyan, "Transparent checkpointing protocol for MPI programs with decentralized initiator", Proceedings of *International Conference on Computer Science and Information Technologies (CSIT2007)*, Yerevan, pp. 227–229, 2007.

[6] Jeffrey Dean, Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", *Sixth Symposium on Operating System Design and Implementation*, San Francisco, pp. 137-150, 2004.