# Multilevel Queue-Based Scheduling for Heterogeneous Grid Environment

**Kumaresh.V.S[#], Prasidh.S[#], Arjunan.B[#], Subbhaash.S[#] and Sandhya.M.K[#]**

**[#] Department of Computer Science and Engineering, Meenakshi Sundararajan Engineering College Chennai, Tamil Nadu, India**

**Abstract**

Grid computing is the federation of pooling resources so as to solve large-scale problems. Scheduling is the main issue in grid computing and is the process of making scheduling decisions over multiple grid resources. In this paper, we propose a scheduling technique which classifies the subtasks based on the priority assigned by the user. This is mainly applicable in places where the high priority critical subtasks may need to be scheduled prior to other low priority subtasks. We thus segregate the subtasks into three queues based on their priority. Subtasks within each queue are reordered based on two new parameters, viz. complexity factor and realization factor, with computational complexity defined as the time of computation of a process. We evaluate the realization factor as the product of number of operations per cycle per processor and the speed of the processor. The subtasks are assigned high priority when both complexity factor and realization factor are high. Once the processes are classified into three queues we make use of a technique similar to round robin that reduces starvation of low and medium priority subtasks. The effectiveness of Starvation free (SF) Scheduling algorithm is evaluated through simulation results.

***Keywords:*** *Grid Computing, Scheduling, Starvation, Multilevel queue, heterogeneous systems.*

## 1. Introduction

The popularity of the Internet and the availability of powerful computers and high-speed networks as low cost commodity components are changing the way we use computers today. These have led to the possibility of using geographically distributed and multi-owner resources to solve large-scale problems in science, engineering and commerce. Recent research on these topics has led to the emergence of a new paradigm known as Grid Computing. There are two advantages of the platform: easy access to powerful computing facilities and effective use of the distributed resources [1].

Computational grids are emerging as a new computing paradigm for solving the new challenging applications in science, technology and engineering. A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities [2]. To achieve the promising potentials of tremendous distributed resources, effective and efficient scheduling algorithms are fundamentally important. Unfortunately, the scheduling algorithms in traditional parallel and distributed systems, which usually run on homogeneous and dedicated resources, e.g. computer clusters, cannot work well in the new circumstances. Hence we propose the new dynamic Starvation Free (SF) Scheduling algorithm.

## 2. Grid Architecture

A grid can be thought of as a distributed system with non-interactive workloads that involve large number of files. A distributed grid comprises of the following components and is represented in Fig1.

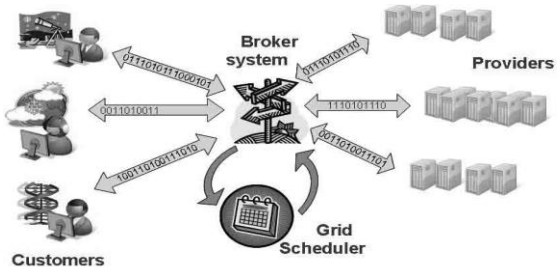



Fig1. Grid Architecture

**Computational grids** involve the intersection of different geographically distributed communities, resource users and the resource-providers. **Grid monitoring** is required by various super-sets of these communities as everyone wants to know something about how the Grid is performing [4].

A **Resource broker** or resource matcher is used to find the memory and complexity of an incoming process which can be a program. It is also responsible for assigning the computing nodes to a task. In other words the Grid Resource Broker (GRB) is a grid portal that allows trusted users to create and handle computational/data grids on the fly exploiting a simple and friendly web-based GUI. GRB provides location-transparent secure access to Globus services, automatic discovery of resources matching the user's criteria, selection and scheduling on behalf of the user.

The **Profiler** in a grid environment can be used by the monitor for predicting the parameters like CPU usage

memory requirement. The **Storage Resource Broker** is a piece of software that sits in between users and resources and provides a storage service by managing users, file locations, storage resources and metadata information [5].

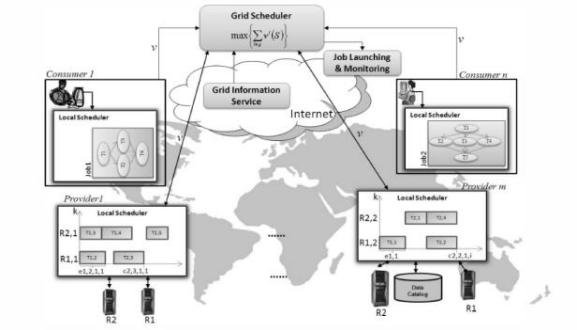



Fig2. Grid Scheduler

Finally comes the **Grid scheduler or grid broker** (Fig2) that makes resource selection decisions in an environment where it has no control over the local resources. The resources are distributed, and information about the systems is often limited or dated [1]. These interactions are also closely tied to the functionality of the Grid Information Services. This Grid scheduling approach has three phases: resource discovery, system selection, and job execution (Fig3). Let's look the proposed scheduling algorithm.

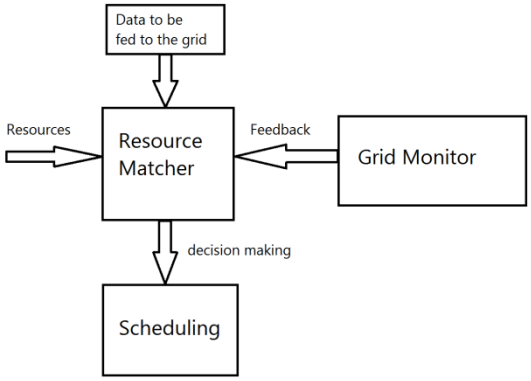



Fig3. Grid Phases

## 3. Related Work

The concept of computational grids, and grid computing in general, is being studied by researchers in many fields, including high-performance computing, networking, distributed systems and web services. The modelling of computational grids with heterogeneous resources is just beginning to be explored [3]. Various algorithms have been proposed to schedule grid resources efficiently. The factors that these algorithms are based on vary as the algorithms themselves. The major factors that have been touched upon until now include time, performance criteria, task priority and realization quotient. All these algorithms do not take user priority into account. The Starvation free scheduling algorithm has shown that by taking user priority into account we can reduce starvation and thus in turn increase the performance of the heterogeneous grid system on the whole.

## 4. Proposed Starvation Free(SF) Scheduling Algorithm

This dynamic (Starvation Free) scheduling algorithm is primarily based on two factors, namely realization factor and computational factor. Our aim is to provide the users of a grid with a starvation free environment and hence we have taken this as our performance metric. The realization factor is calculated by considering the work factor, which is dependent on two parameters namely, number of operations per cycle and speed of the processor. This makes sure that the subtasks are assigned to the resources efficiently. The complexity factor is computed by considering the waiting time and the computational complexity of the subtasks in addition to the priority assigned by the user.

The inclusion of waiting time in the proposed algorithm is for the purpose of reducing the probability of occurrence of starvation, thus enabling a good and efficient grid environment. Both these factors are assigned with three types of values namely, High, Medium and Low for each of the subtasks. The overall priority of a subtask is obtained by combining both these factors in such a way that the one with both the factors as High is assigned the higher priority and the one with both the factors as Low is assigned the lower priority. After prioritizing, when a collision occurs among subtasks with same priority, we consider the one with higher user priority to be assigned to a competing resource. If collision still exists, then the one with the greater waiting time is considered and if collision still persists, then the subtask with the better computational complexity is chosen.

We now maintain three levels of queues for each of the subtasks with user priority High, Medium and Low and sort them with the above assigned priority. Now the subtasks are assigned to the resources in a round-robin fashion by selecting three subtasks at once from the High priority queue along with two other subtasks from the Medium priority queue and one subtask from the Low priority queue. This ensures that subtasks with high user priority are given higher preference over than the ones with lesser priority and care is taken to provide a starvation free environment. The algorithm is split into three modules as follows.

**STARVATION FREE (SF) Scheduling Algorithm**

MODULE 1: Realization Factor assignment

For each of the resource available in the grid, a Work Factor (WF) is assigned first, which is the product of the number of operations per cycle and the speed of the processor. Then the Min, Max and Mid ranges of this factor are chosen by the user so that the subtasks are assigned with three types of values namely, High, Medium and Low as their Realization Factor (RF).

```
Assign Realization_Factor (ResourceList RL [])
  While (RL []! =NULL)
        For each RL[i] in RL []
            WF[i] =OC*SP
 /*WF=Work Factor
   OC = No. of operations per cycle per
processor
  SP = Speed of the processor */
 End while
Find the Max, Min & Mid ranges in WF []
  For each subtask in WF []
        If WF[i]>=Max
        RF[i] =High
 /*RF=Realization Factor*/
        Else if WF[i]>=Mid
        RF[i] =Medium
        Else
        RF[i] =Low
        End if
  End for
End Assign Realization_Factor
```

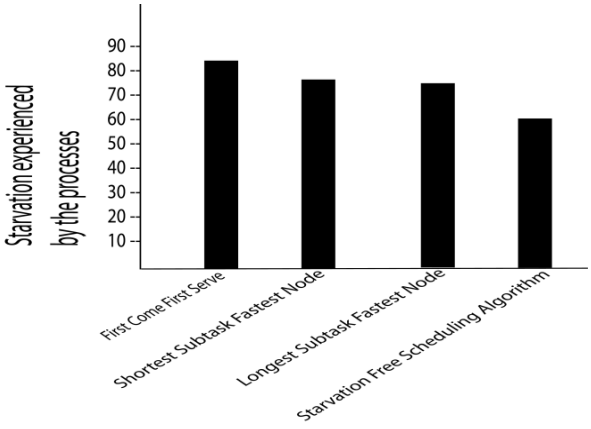
MODULE 2: Complexity Factor assignment
This subroutine considers two parameters namely, the waiting time and the computational complexity of the subtasks. The waiting time of the subtasks is multiplied with a suitable coefficient 'i' (1, 2 or 3) according to the value of user priority (Low, Medium or High). Then the Waiting time- Computational complexity (WC) factor for each subtask is computed by adding computational complexity with the coefficient and multiplying the waiting time of the subtasks. Further the Min, Max and Mid ranges of this factor are chosen by the user so that the subtasks are assigned with three types of values namely, High, Medium and Low as their Complexity Factor (CF).

```
Assign Complexity_Factor (LocalList LL [])
  While (LL []! =NULL)
        For each subtask
        If (User_Priority[i] ==High)
      WC[i] =3*Waiting_Time[i] + Comp[i]
/*Comp[i] = Computational Complexity of the
subtask*/
        Else if (User_Priority[i] ==Medium)
     WC[i] =2*Waiting_Time[i] + Comp[i]
        Else
     WC[i] =Waiting_Time[i] + Comp[i]
        End if
   End while
Find the Max, Min & Mid ranges in WC []
    For each subtask in WC []
        If (WC[i]>=Max)
        CF[i] =High
    /*CF=Complexity Factor*/
        Else if (WC[i]>=Mid)
        CF[i] =Medium
        Else
   CF[i] =Low
   End if
End for
End Assign Complexity_Factor
```

MODULE 3: Calculation of Starvation Free priority
This takes into account the values of Realization Factor (RF) and Complexity Factor (CF) for each of the subtask. The Starvation Free Priority (SF_Priority) for a subtask is assigned 1 (Highest) when both RF and CF are High and 9(Lowest) when both RF and CF are Low. Thus the subtasks are sorted according to the value of SF_Priority in the three levels of user-priority queues and allocated resources by the scheduler in a round-robin fashion.

```
Assign SF_Priority (Local_List LL [])
  While (LL []! =NULL)
     For each subtask
If (CF[i] ==High AND RF[i] ==High)
    SF_Priority[i] =1
    /*SF_Priority=Starvation Free Priority*/
Else If (CF[i] ==High AND RF[i] ==Medium)
     SF_Priority[i] =2
Else If (CF[i] ==High AND RF[i] ==Low)
     SF_Priority[i] =3
Else If (CF[i] ==Medium AND RF[i] ==High)
     SF_Priority[i] =4
Else If (CF[i] ==Medium AND RF[i] ==Medium)
      SF_Priority[i] =5
Else If (CF[i] ==Medium AND RF[i] ==Low)
      SF_Priority[i] =6
Else If (CF[i] == Low AND RF[i] ==High)
      SF_Priority[i] =7
Else If (CF[i] == Low AND RF[i] == Medium)
      SF_Priority[i] =8
Else If (CF[i] ==Low AND RF[i] ==Low)
      SF_Priority[i] =9
End if
End while
End Assign SF_Priority
```

## 5. Performance Study



**First Come First Serve (FCFS):**
This scheduling algorithm schedules the subtask on a "First Come First Serve" basis. This algorithm is simple and is not based on any of the factors like complexity factor and realization factor. It shows very low computation results when compared to other scheduling algorithms since it assigns resources for subtasks in the order of their arrival.

**Shortest Subtask Fastest Node (SSFN):**
The Shortest Subtask Fastest Node algorithm assigns the subtask with a smaller value of computational complexity to the fastest node available in the resource. Shortest Subtask Fastest Node is a scheduling algorithm, which tries to reduce the overall turnaround time of the subtask. SSFN is more stable in handling subtask and hence outperforms FCFS scheduling algorithm.

**Longest Subtask Fastest Node (LSFN):**
The scheduling algorithm, commonly used for the assignment of complex subtasks to high efficiency resources is the Longest Subtask Fastest Node (LSFN) algorithm. Though there is room for improvement, it tries to reduce the overall execution time of the subtask. From the LSFN algorithm we can infer that LSFN outperforms FCFS and the SSFN as the subtasks are assigned to faster nodes in the resource which leads to shorter execution time. All the above mentioned scheduling algorithms lead to starvation of low and medium priority resources whereas the proposed algorithm does not.

**SF Scheduling algorithm:**
The main focus of this algorithm is to provide a starvation-free grid environment. This is done by considering the waiting time as another main parameter in the scheduling process in addition to the user-priority and computational complexity of the subtasks. Thus the subtasks with high values of waiting time, user-priority and computational complexity are allocated to the fastest resource(s) first, which prevents the problem of indefinite waiting. After being assigned with priorities, the subtasks are now put up in three different levels of queues namely, High, Medium and Low user-priority. The scheduler is made to schedule the subtasks in such a way that these subtasks are allocated to the available resources in a round-robin fashion. For instance, if there are six resources available at a time in the grid and there are a long list of subtasks waiting for allocation; then the SF algorithm is applied to classify these subtasks in three levels of queues. Then the subtasks are assigned to the six available resources by choosing three from the High user-priority queue, then two more form the Medium user-priority queue and finally one subtask from the Low user-priority queue. Thus the performance is really high here when compared to its predecessors.

## 6. Conclusion

Multi-level queue based scheduling mechanism is put forward along with the algorithm for reducing starvation. The proposed Starvation Free (SF) Scheduling algorithm can be applied widely and it helps in scheduling resources efficiently, resulting in a starvation free grid environment. Results from simulation experiments demonstrate that the algorithm optimizes the resource nodes and resource utilization rate gets a substantial increase. The multi-level queue based scheduling algorithm for the heterogeneous grid environment has high performance as compared to the other scheduling algorithms for the grid environment. In the future, our algorithm should be improved to be able to broadcast and move waiting jobs into execution as soon as resource has completed jobs, without having to wait for the next scheduling event.

## References

[1] Amir M Bidgoli, Zahra Masoudi Nezad: A new scheduling algorithm design for grid computing tasks, 5th Symposium on Advances in Science and technology, Iran.
[2] G.Sumathi, S.Sathyanarayanan, R.Santhosh Kumar "MidSFN Local Scheduling for Heterogeneous Grid Environment", IJCSI, Vol.9, Issue 3, No.3, May 2012.
[3] Fangpeng Dong and Selim G. Akl, Scheduling Algorithms for Grid Computing: State of the Art and Open Problems
[4] Cho-Chin Lin and Chun-Wei Shih, An Efficient Scheduling Algorithm for Grid Computing with Periodical Resource Reallocation.
[5] Economic-Based Modeling for Resource Scheduling in Grid Computing, Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design.

**Kumaresh V S** is pursuing final year B.E. Computer Science and Engineering in Meenakshi Sundararajan Engineering College, Chennai, Tamil Nadu, India. He is a member of CSI and also an Oracle Certified Professional - Java SE 6 Programmer. His research interests include Scheduling in Grid, Design, Analysis and Optimization of Scheduling algorithms and Stream Analytics in Cluster Technology.

**Prasidh S** is pursuing final year B.E. Computer Science and Engineering in Meenakshi Sundararajan Engineering College, Kodambakkam, Tamil Nadu, India. He is a member of IEEE, CSI and a Microsoft Student Partner. His research interests include Cloud Computing, Big Data analytics, Parallel and Distributed Systems.

**Arjunan B** is pursuing final year B.E. Computer Science and Engineering in Meenakshi Sundararajan Engineering College, Kodambakkam, Tamil Nadu, India. He is a member of CSI. His research interests include Databases, Data mining and Grid Computing.

**Subbhaash S** is pursuing final year B.E. Computer Science and Engineering in Meenakshi Sundararajan Engineering College, Kodambakkam, TamilNadu, India. He is a member of CSI. His research interests include Grid Scheduling and Cloud Computing.

**Sandhya MK** received her Bachelor of Engineering in Computer Science and Engineering from University of Madras and Master of Engineering in Computer Science and Engineering from Anna University. She is working as Assistant Professor in the Department of Computer Science and Engineering in Meenakshi Sundararajan Engineering College. She is currently pursuing research in security issues in wireless sensor networks at Anna University. She is a Life member of ISTE.