Task Scheduling Strategies for Dynamic Reconfigurable Processors in Distributed Systems

M. Faisal Nadeem, S. Arash Ostadzadeh, Stephan Wong, and Koen Bertels
Computer Engineering Laboratory, Delft University of Technology, The Netherlands
{M.F.Nadeem, S.A.Ostadzadeh, J.S.S.M.Wong, K.L.M.Bertels}@TUDelft.nl

ABSTRACT

Reconfigurable processors in distributed grid systems can potentially offer enhanced performance along with flexibility. Therefore, grid systems, such as TeraGrid, are utilizing reconfigurable computing resources next to general-purpose processors (GPPs) in their computing nodes. In general, the application task scheduling largely affects the near-optimal performance of resources in distributed grid systems. The inclusion of reconfigurable nodes in such systems requires to take into account reconfigurable hardware characteristics, such as, area utilization, reconfiguration time, and time to communicate configuration bitstreams, execution codes, and data. Generally, many of these characteristics are not taken into account by traditional task scheduling systems in distributed grids. In this paper, we present a simulation framework for application task distribution among different nodes of a reconfigurable computing grid. Furthermore, we propose three different task scheduling strategies, namely Optional Closest Match (OCM)*, Exact Match Priority* (EMP)*, and Sufficient-Area Priority (SAP). The simulation results are presented based on the* average scheduling steps *required by the scheduler to accommodate each task, the* total scheduler workload*, and the* average waiting time per task*. We compare the impacts of the three scheduling strategies on these metrics. In addition, we present a thorough discussion of the results. In particular, the results show that the two key metrics* average scheduling steps per task *and* average waiting time per task *are reduced for the EMP and the SAP when compared to the OCM.*

KEYWORDS: Distributed systems; Reconfigurable computing; Simulation framework; Task scheduling; Resource management.

1. INTRODUCTION

Distributed Computational grids provide inexpensive and dependable access to high-end computational resources that are geographically dispersed over the world [1]. Some notable worldwide projects in grid computing are Globus [2], Legion [3], and Unicore [4]. In the recent years, reconfigurable computing is utilized more commonly in high-performance computing, such as, multimedia processing, bioinformatics, and cryptography [5]. The reason is that reconfigurable devices provide both increased performance without expensing on flexibility, i.e., allowing quick changes between the supported applications. Consequently, distributed computing networks (such as TeraGrid [6]) have incorporated reconfigurable computing resources next to general-purpose processors (GPPs) in their computing nodes. Proposals to better exploit the characteristics of reconfigurable computing merged with the requirements of computational grids have been suggested, e.g., Collaborative Reconfigurable Grid Computing (CRGC) in [7].

Scheduling algorithms have been thoroughly studied in conventional parallel and distributed systems. Various state-of-the-art scheduling algorithms for traditional grids have been proposed in [8] and [9]. The near-optimal utilization of grid resources predominately depends on application task scheduling onto the computing nodes. Therefore, scheduling algorithms are of paramount significance for computational grids and new ones must be developed when the node characteristics are changed. The inclusion of reconfigurable processors in the computational nodes requires the rethinking of existing scheduling techniques in order to take into account reconfigurable processor characteristics, such as, area utilization, reconfiguration time, (possible) performance increase, and the time required to transfer configuration bitstreams, execution codes, and data.

In this work, we present the design of a simulation framework for the application task scheduling for reconfigurable processors in a distributed grid system. The proposed design can model complex reconfigurable computing nodes, processor configurations, and tasks along with GPPs. The processor configurations can be modeled using a number of different reconfiguration parameters, such as, area required, reconfiguration time delay, network delay, etc. Various parameters (e.g., node area range, task arrival rate, reconfiguration time range, etc.) can be set according to the requirements of a particular simulation. The simulation framework generates a report on different result statistics at each simulation run.

Using our simulation framework, we propose and implement three different task scheduling strategies, namely *Optional Closest Match* (OCM), *Exact Match Priority* (EMP), and *Sufficient-Area Priority* (SAP). The scheduling strategies are described in detail and the results show the impacts of each scheduling strategy on three different system metrics *total scheduling steps* required by the scheduler to accommodate each task, *average waiting time per task* and the *total workload of the scheduler*. In particular, the results show that the two key metrics *average scheduling steps per task* and *average waiting time per task* are reduced for the EMP and the SAP when compared to the OCM.

The remainder of the paper is organized as follows. Section 2 provides related work. The background and scope are discussed in Section 3. We discuss the design of the our simulation framework in Section 4. Three different scheduling strategies are proposed and discussed in Section 5. The simulation environment and results are explained in Section 6. Finally, we provide our conclusions and future work in Section 7.

2. RELATED WORK

Many state-of-the-art simulators for task scheduling in distributed systems are proposed. Few examples are SimGrid [10], MicroGrid [11], and GridSim [12]. These simulators can model computational resources composed of GPPs and there is no specific simulator to take into account the behavior of reconfigurable nodes in distributed computing environment. In [7], CRGridsim has been proposed by extending Gridsim [12] to add reconfigurable elements in grids. However, it only considers speedup factor of a reconfigurable processor over a GPP, as the reconfiguration parameter. Other important parameters, such as reconfigurability, reconfiguration delay, reconfiguration method (partial or full), area utilization, and hardware technology have not been taken into account. Most of the work [8] [9] on scheduling algorithms has been proposed for traditional grid networks with processors of heterogeneous computing capacity, but they are not reconfigurable processors.

3. BACKGROUND AND SCOPE

Task scheduling in grids is defined as scheduling *n* tasks with a given execution time onto *m* resources. In this section, we formulate the scheduling problem and discuss the models for task, processor configuration, and nodes. A typical reconfigurable node is represented by the following tuple:

$$Node_i\,(NID, Area, \mathrm{C}_{existing}, state) \qquad (1)$$

Where NID represents the *node number*, $Area$ the total reconfigurable area of node i and $C_{existing}$ the current processor configuration on the node. Initially, no configuration is assumed on a computing node and, therefore, set to C_{blank}. Furthermore, $state$ represents the status of the node i; whether it is busy or idle. Subsequently, the general configuration of a processor to be configured on a node, is represented as:

$$\mathrm{C}_i\,(CID, Area, P_{type}, parameters) \qquad (2)$$

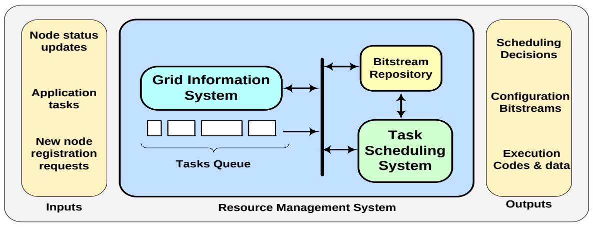
Where CID represents the *configuration number* and $Area$ is the total reconfigurable area required by configuration i . P_{type} represents the processor type required by tasks in the system. $parameters$ is a list of attributes of a particular processor which provide the details of the P_{type} processor required by tasks. Typical examples of P_{type} are soft-core processors, digital signal processors (DSPs), custom computing units (CCUs), etc. For instance, a soft-core ρ-VEX VLIW processor (specified by P_{type}) implemented on an FPGA, can be adopted to several $parameters$ such as, the number of issue slots, cluster cores, the number and types of functional units, or the number of memory units [13]. Similarly, the following tuple represents an application task:

$$Task_i\,(TID, t_{required}, C_{pref}) \qquad (3)$$

Where TID represents the *task number* and $t_{required}$ is the execution time required by task i if it is processed on its preferred processor configuration (C_{pref}). C_{pref} is the preferred processor configuration demanded by task i. This configuration can be a specific processor that can be implemented on a reconfigurable node. Based on above definitions, we formulate the problem as follows.

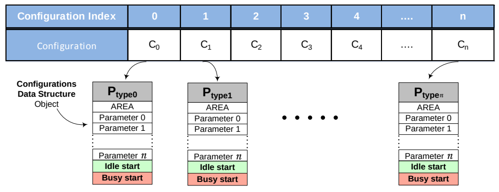
Given a set of reconfigurable nodes represented by (1) and a set of tasks defined by (3), find a scheduling algorithm to map all tasks onto nodes, following a particular strategy to minimize key system metrics, such as waiting time, total reconfigurable area waste, and scheduling workload etc. In this work, we discuss three different scheduling strategies to investigate their effects on these metrics.

In a distributed system with reconfigurable computing nodes, each node consists of a reconfigurable processor of fixed area and it is configured with a particular processor configuration C_i. Furthermore, the network contains a *Resource Management System* (RMS) which takes in user application tasks and implements a scheduling algorithm to

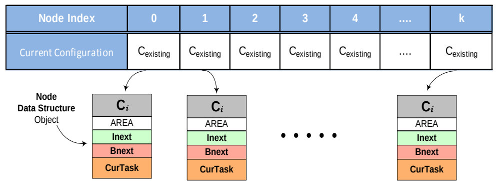


(a)

Figure 1: The Resource Management System for a Distributed System with Reconfigurable Nodes.



(a) The Configurations List.



(b) The Nodes List.

Figure 2: The Proposed Data Structures.

map tasks onto nodes. The RMS, depicted in Figure 1a, consists of a *Grid Information Service* (GIS), which is a dynamically changing module. It contains information about the current states of all the nodes in the network. This information consists of current state (busy or idle) of a particular node, its current C_i configuration and its available reconfigurable area. The status information of each node is updated each time the current state of that node is changed. The core of the RMS is a task scheduler which implements a scheduling algorithm to assign tasks onto nodes. Moreover, the RMS contains a *bitstream repository* which consists of a set of bitstreams of different configurations of C_i. These bitstreams are already synthesized, implemented and tested on all the nodes and are considered as *valid bitstreams* for the system. Each application task in the grid requires a preferred processor configuration (C_{pref}) with specific $parameters$. The parametrization characteristics of C_i processor can be exploited by the grid system by either sending the required $parameters$ of the C_{pref} of a task or by sending the corresponding bitstream of the C_{pref} to the respective reconfigurable computing nodes.

4. SIMULATION FRAMEWORK DESIGN

In this section, we present the design of the proposed simulation framework. We briefly discuss the proposed data structures and different modules in the framework.

4.1. Data Structures

Here we discuss the required data structures developed for resource management in our proposed simulation framework. These proposed data structures are used to maintain the states of the nodes in the resource management system, depicted in Figure 1a. All possible configurations in the grid are stored in a data structure which is called the *configurations list* depicted in Figure 2a. Each entry in this list corresponds to the data structure to provide the details of C_i configuration and its $parameters$ as shown in tuple (2). Furthermore, it contains two pointers *Idle start* and *Busy start*. The *Idle start* points to the first node of the linked list of the idle nodes with the corresponding configuration of a particular node. Similarly, the *Busy start* points to the first node of the linked list of busy nodes with the corresponding configuration of a particular node. Similarly, Figure 2b depicts the *Node* data structure which represents a typical grid node. It contains the current or the existing C_i configuration of the node, reconfigurable area, pointers to the next node in the idle list with the same configuration (*Inext*), pointer to the next node in the busy list with the same configuration (*Bnext*), and a pointer to the current task being executed on the node (*CurTask*). The node structure is represented by tuple (1). In the following sections, we describe how these data structures are utilized by the simulation framework and the scheduling module.

4.2. The Simulation Framework

Figure 3 depicts our proposed simulation framework designed for the implementation of the required data structures and task scheduling process. It generates synthetic tasks, grid nodes, and different node configurations. The *Synthetic Task Generator* generates tasks based on a given task distribution function. The *Node Generator* module generates grid nodes with different reconfigurable area sizes. A user can set the upper and lower area ranges of the nodes for simulation purpose. For a realistic study, these area ranges can be set according to the actual area sizes of the reconfigurable devices available. Similarly, the *Configuration Generator* module generates a variety of processor configurations (C_i) for a particular grid node.

The tasks, nodes, and configurations generator modules use random number generation methods which are based on the *Ziggurat Method* [14] using the algorithm presented in [15] for generating *Gamma variables*. Various random number

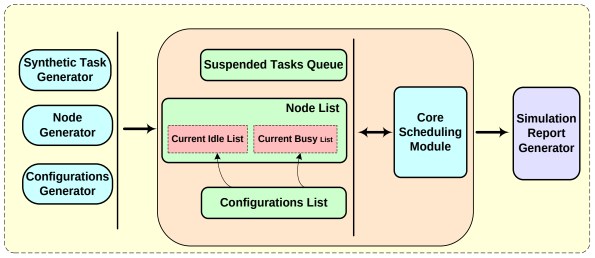


Figure 3: The Simulation Framework.

distributions, such as *Poisson*, *Binomial*, *Gamma*, *Uniform random* etc., are provided by a random number generator class. The tasks arrival times correspond to the statistical distributions available in the class. The core of the simulation framework implements a scheduling module which implements different scheduling strategies, as discussed in the Section 5. Furthermore, the framework implements the data structures described in Section 4.1. The user-defined configurations are stored in the *Configuration list* and the linked lists are implemented accordingly. The *Suspension Queue* holds $TIDs$ of the tasks which can not be accommodated by any node upon their arrivals. This particular situation occurs, when all the nodes with C_{pref} are busy and no blank or idle nodes (with sufficient area) are available for executing the current task. The suspension queue is checked automatically after the completion of each running task to find a potential replacement. The *Core Scheduling Module* implements different scheduling strategies and the methods required to interact with the node and configuration data structures.

Finally, the *Simulation Report Generator* module stores the statistics gathered during the simulation process. It reports on different performance metrics, such as *number of scheduling steps* required by the scheduler to accommodate each task, *total scheduler workload*, *average task waiting time*, *average task running time*, *reconfiguration count per node*, *reconfiguration time per task,* etc. *The average number of scheduling steps required per task* is an indication of the total number of search links explored by the scheduler before a task can be assigned to a proper node in the system. This metric closely reflects the quantitative value of the time taken by the scheduler to accommodate a task. The *total scheduler workload* provides an estimate of the workload required by the scheduler to assign a task to a proper node and to perform various housekeeping jobs, such as initialization of node and configuration lists, updating the linked lists during task assignments, suspension queue checking, etc. These performance metrics are important because they are proportional to the system time of a host executing the scheduler, hence, critical in comparing different scheduling strategies. The *average waiting time per task* provides the average time elapsed from the time a task is submitted to the system until the time it is assigned to a node to be executed.

Algorithm 1 The *Optional Closest-Match* Scheduling

```
Initialize the configs list
Initialize the nodes list
Begin TaskSchedule (Task CT)
Phase-I
- If possible, allocate the CT to the best-match node configured with its $C_{pref}$.
- Otherwise if possible, allocate CT to a blank node after configuring it with $C_{pref}$ of CT.
- Otherwise, allocate CT to any idle node (currently configured with a $C_{existing}$) after reconfiguring it with $C_{pref}$ of CT.
Phase-II
- If CT required a $C_{pref}$ not available in the configs list, if possible, allocate CT to any available node with closest-match (of $C_{pref}$) configuration.
- Otherwise, put CT in suspension queue if at least one busy node with sufficient area is available.
- Otherwise, discard CT.
```

5. SCHEDULING STRATEGIES

In this section, we describe the three different scheduling strategies used in the core scheduling module in the simulation framework. The scheduling module takes tasks, configurations, and nodes as inputs and assigns the tasks onto different nodes, based on a particular scheduling strategy.

5.1. *Optional Closest-Match* (OCM) Strategy

An incoming task (current task-*CT*) is assumed to have its preferred configuration (C_{pref}) as shown in the tuple (2). The algorithm assigns the *CT* onto a grid node of its C_{pref}, if possible. Initially, the C_{pref} of the *CT* is matched in the *configurations list* depicted in Figure 2a. If the C_{pref} is found in the list, then the algorithm looks for a suitable idle node of the same configuration by traversing the list of idle nodes of that particular configuration.

If idle nodes are available, the *CT* is allocated to the *best-match* node. The node with the *best-match* is one which makes minimum reconfiguration area waste among all the nodes in the linked list. On the other hand, if C_{pref} is found in the *configurations list*, but there is no idle node available, then the scheduler searches for the list of blank nodes (which are not yet configured). If neither idle nor blank nodes are available with the C_{pref}, then the scheduler searches for any potential idle node of any other configuration, by traversing the *configurations list*. Subsequently, a node with the minimum sufficient area is selected and the task is allocated to it after making it a blank node and reconfiguring it with C$_{pref}$ of the *CT*.

In case the C$_{pref}$ of the *CT* is not found in the configurations list (phase-II in Algorithm 1), the scheduler searches for a node with the *closest-match* configuration to the C$_{pref}$ of *CT*. The *closest-match* processor configuration is the one which can still execute the *CT* but takes more time for processing than the *exact-match* configuration. Once the *closest-match* configuration is found, the scheduler looks for suitable idle nodes with that configuration. If not found, it looks for blank nodes with sufficient area to configure the *closest-match* configuration and accommodate the *CT*. If blank nodes are also not found, then the scheduler puts the task in the suspension queue to wait for any busy node with sufficient area to become idle. Finally, a task is discarded if no node with sufficient reconfigurable area can be found to accommodate the current task.

5.2. *Exact-Match Priority* (EMP) Strategy

The EMP strategy is described in Algorithm 2. At the end of Phase-I, if the scheduler can not find a suitable node to accommodate the *CT*, we prefer to wait for a potential node with C$_{pref}$ to be idle than to go for a *closest-match* configuration. Therefore, the scheduler puts the task in the suspension queue. When the queue is checked, the *CT* is assigned to the first *recently released* node which is configured with C$_{pref}$ of the *CT*.

Algorithm 2 The *Exact-Match* Priority Scheduling

```
Initialize the configs list
Initialize the nodes list
Begin TaskSchedule (Task CT)
Perform Phase-I in the Algorithm-1
- If there is at least one busy node with C_pref configuration
and sufficient area, put the CT in the suspension queue.
Perform Phase-II in the Algorithm-1
```

5.3. *Sufficient-Area Priority* (SAP) Strategy

Algorithm 3 describes the SAP scheduling strategy. The main difference between EMP and SAP is that in SAP, we relaxed the restriction of accommodating the suspended tasks on the queue with the nodes of the same configuration. In other words, the scheduler may select any node regardless of its configuration to accommodate a task on the suspension queue provided that there is enough area. At the end of Phase-I, if the scheduler can not find a suitable node, it puts the *CT* in the suspension queue. When the queue is checked, the allocation of *CT* is made to the first *recently released* node with sufficient area to accommodate the task. If the node is already configured with C$_{pref}$, the task is allocated to it. Otherwise, the node is first configured with C$_{pref}$ and then *CT* is allocated to it.

Algorithm 3 The *Sufficient-Area* Priority Scheduling

```
Initialize the configs list
Initialize the nodes list
Begin TaskSchedule (Task CT)
Perform Phase-I in the Algorithm-1
- If there is at least one busy node with sufficient area, put
the CT in suspension queue.
Perform Phase-II in the Algorithm-1
```

During the scheduling process in all strategies, the corresponding busy or idle lists are updated accordingly when the *CT* is allocated to a certain node. If the scheduler selects a blank node to accommodate the *CT,* the node is removed from the blank list and is attached to the list of busy nodes.

6. SIMULATION ENVIRONMENT AND EXPERIMENTAL RESULTS

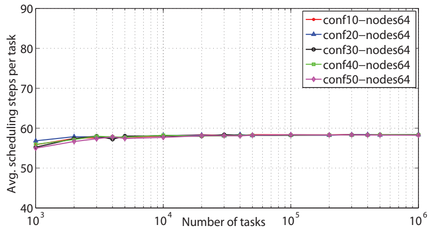
The main simulation parameters used in our experiments are presented in Table 1. The experiments were conducted on different sets of tasks consisting of 10^3 to 10^6 tasks. Completion time required by tasks ranges between [100...10000] timeticks. Similarly, the C_{pref} for any given task requires area within range [100...2500] area units (e.g., area slices or FPGA LUTs). For 10 % of the total tasks, we assign a preferred configuration (C$_{pref}$) that can not be found in *configurations list.* Therefore, these tasks are assigned to the reconfigurable nodes with *closest-match* configuration by the scheduler at run time.

Table 1: The Main Simulation Parameters.

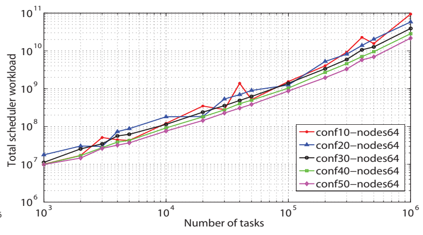
Parameter	Value
Total number of tasks generated	[$10^3 \ldots 10^6$]
Total number of nodes	64, 128
Total number of configurations	10, 20, 30, 40, 50
Next task arrival interval	[1...50]
Configurations required area range	[100...2500]
Node available area range	[1000...5000]
Task required timeslice range	[100...10000]
Reconfiguration time range	[1...5]

6.1. Results Discussion

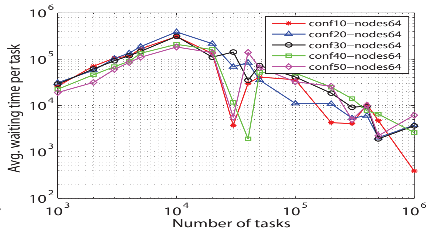
All the experiments were performed on Intel Core 2 Duo CPU E8400@3.00GHz, 64-bit machine running Linux v.2.634. Simulation experiments were conducted for all 3 scheduling strategies, in order to compute the *total scheduler workload*, *average number of scheduling steps required per task*, and *average waiting time per task.* In the



(a) Avg. scheduling steps per task.

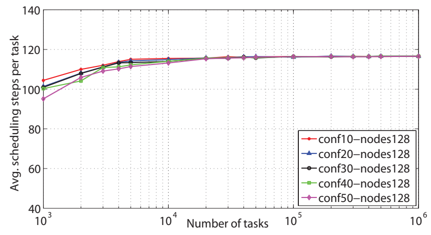


(b) Total scheduler workload (y-axis on log scale).

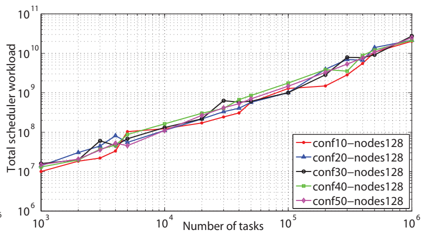


(c) Avg. task waiting time (y-axis on log scale).

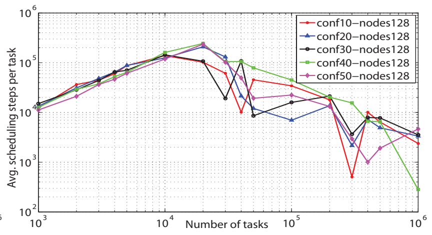
(A) Total Number of Nodes=64.



(d) Avg. scheduling steps per task.



(e) Total scheduler workload (y-axis on log scale).



(f) Avg. task waiting time (y-axis on log scale).

(B) Total Number of Nodes=128.

Figure 4: The *OCM* Scheduling Strategy with a Fixed Set of Configurations. (A) Total nodes=64 and (B) Total nodes=128.

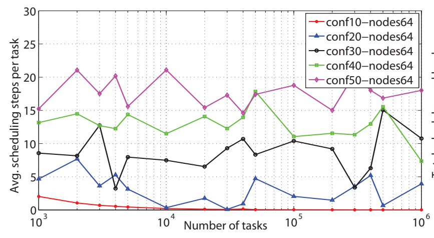
following sections, we discuss results for each scheduling strategy in detail. Since we used the uniform random numbers to generate large number of tasks, the average values remain the same for several run of simulations for each experimental setup.

The *OCM* strategy. Figure 4 depicts simulation results for the OCM scheduling strategy. Figures 4 (A) and 4 (B) give *average scheduling steps per task*, *total scheduler workload*, and *average task waiting time* results for 64 and 128 nodes, respectively.

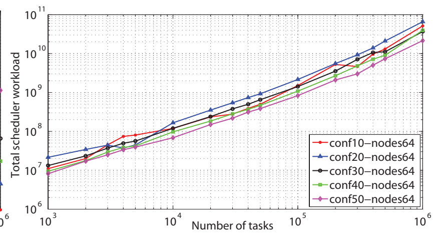
Figures 4a and 4d compare the *average scheduling steps per task* for 64 and 128 nodes, respectively. The scheduling steps in both cases are less for smaller number of tasks (10^4 or less) and then, they reach to a steady-state. This occurs because, initially, the corresponding linked-lists for configurations are not formed. Consequently, the number of scheduling steps are less. Later, when the linked-lists are populated with corresponding configurations and due to more workload stress on the system, the lists most likely are searched completely to accommodate an incoming task. Although an option of sending a particular task to a *closest-match* configuration node is available, but still the number of nodes are too scarce to make use of *closest-match* option, resulting in putting more tasks in the suspension queue. In the case of 128 nodes, the *average scheduling steps per task* are relatively high (around 120) due to more blank nodes being searched initially for an *exact-match* configuration. As depicted in Figures 4b and 4e, the *total scheduler workload* in both cases, increases exponentially (y-axis are in logarithmic scale) as the number of tasks grow showing the workload burden on the scheduler. Due to rapid arrival rate and insufficient nodes, the tasks are put more frequently into suspension queue, which increases scheduler burden. It is slightly more for lesser number of nodes (64 nodes) due to more congestion in the suspension queue. Similarly, due to more frequent use of the suspension queue, the *average task waiting time* is also very high (see figures 4c and 4f).

The *EMP* strategy. Figure 5 depicts simulation results for the EMP scheduling strategy. Figures 5 (A) and (B) give *average task scheduling steps*, *total scheduler workload*, and *average task waiting time* results for 64 and 128 nodes, respectively.

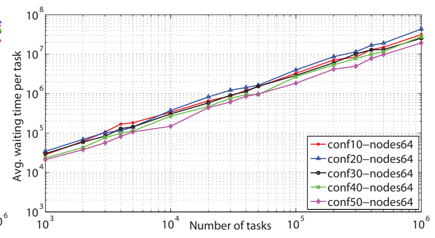
In this strategy, after phase-I (see algorithm 2), the tasks searching for an *exact-match* node (90 percent of total tasks in this case) are put into suspension queue instead of looking for a *closest-match* option. Later, when the queue is checked at each *recently released* node, a suspended task is allocated to the node only if it has the same C_{pref} configuration. As expected, this strategy puts more burden on the suspension queue process which is clearly depicted in the higher waiting time metric (see figures 5c and 5f). The *total scheduler workload* is similar to the OCM, but the *average scheduling steps per task* decreases (see figures 5a and 5d) because the scheduler never enters into the phase-II for 90 percent of tasks looking for *exact-match* nodes as it never goes through the *closest-match* option for those tasks.



(a) Avg. scheduling steps per task.

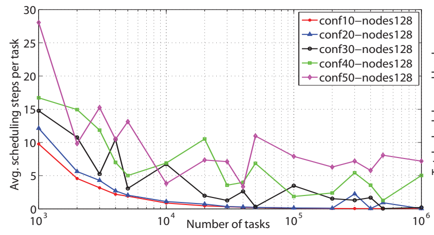


(b) Total scheduler workload (y-axis on log scale).

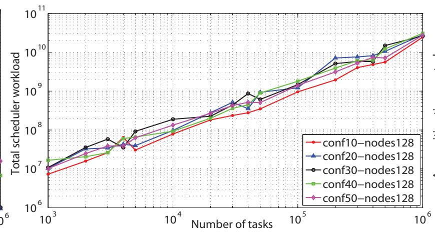


(c) Avg. task waiting time (y-axis on log scale).

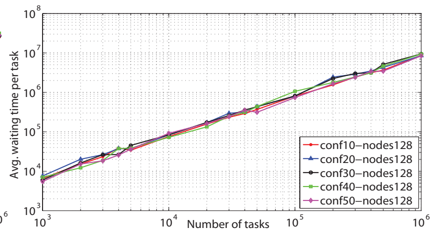
(A) Total Number of Nodes=64.



(d) Avg. scheduling steps per task.



(e) Total scheduler workload (y-axis on log scale).



(f) Avg. task waiting time (y-axis on log scale).

(B) Total Number of Nodes=128.

Figure 5: The *EMP* Scheduling Strategy with a Fixed Set of Configurations. (A) Total nodes=64 and (B) Total nodes=128.

The *SAP* strategy. Figure 6 depicts simulation results for the SAP scheduling strategy. Figures 6 (A) and (B) give *average task scheduling steps*, *total scheduler workload*, and *average task waiting time* results for 64 and 128 nodes, respectively.

This strategy is similar to the EMP, but the condition on checking the suspension queue is relaxed by allocating a suspended task to a *recently released* node if it has sufficient area to accommodate that task. It clearly decreases the *total scheduler workload* and *average task waiting time* for a given number of tasks. The *average scheduling steps per task* metric is similar to the EMP strategy. *Average task waiting time* is considerably less as compared to exact-match strategy because the scheduler assigns the tasks to any node with sufficient area.

From all sets of experiments, it can be concluded that the system behavior highly depends on the number of nodes, tasks arrival rate, and the scheduling strategy adopted by the scheduler. Important system metrics such as *average scheduling steps*, *total scheduler workload*, and *average waiting time per task* behave in an expected manner for each scheduling strategy and given sets of nodes, configurations and tasks. Furthermore, it can be concluded that the *average task waiting time* can be reduced considerably by using the SAP strategy for a given set of parameters.

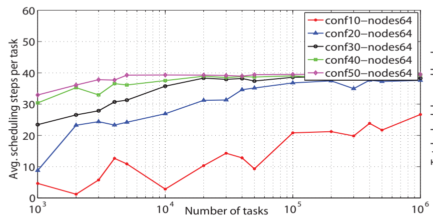
7. CONCLUSIONS AND FUTURE WORK

We presented a simulation framework to analyze task scheduling strategies in a distributed grid system utilizing reconfigurable nodes. The proposed simulation framework can be used to investigate the desired system scenario(s) for a particular scheduling strategy and a given number of tasks, grid nodes, configurations, task arrival distributions, area ranges, and task required times etc. Subsequently, it can be utilized for implementing several other scheduling strategies to optimize individual or some system metrics. Furthermore, we proposed and implemented three different task scheduling strategies. The simulation results for the proposed strategies were discussed, based on *average scheduling steps required per task*, *total scheduler workload*, and *average waiting time per task*. The results show the impact of these strategies on key system metrics. In our future work, we will extend the proposed framework to investigate different configuration strategies such as partial reconfiguration techniques and their effect on different system metrics.

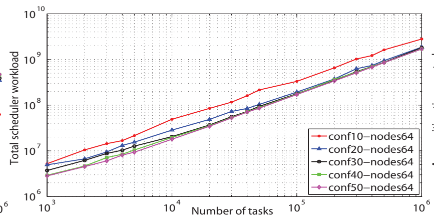
REFERENCES

[1] I. Foster and C. Kesselman, "Computational Grids," in *Selected Papers and Invited Talks from the 4th International Conference on Vector and Parallel Processing (VECPAR)*, pp. 3–37, 2001.

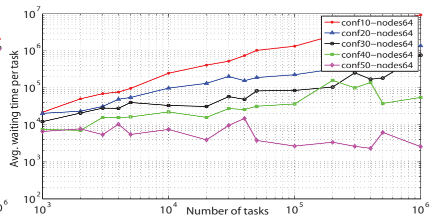
[2] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal*



(a) Avg. scheduling steps per task.

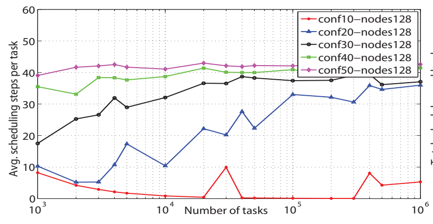


(b) Total scheduler workload (y-axis on log scale).

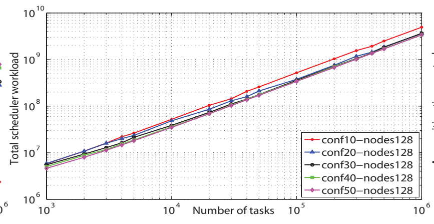


(c) Avg. task waiting time (y-axis on log scale).

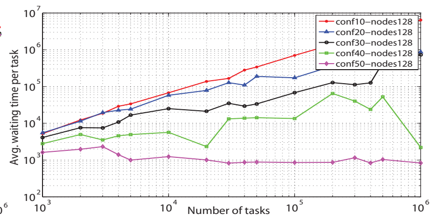
(A) Total Number of Nodes=64.



(d) Avg. scheduling steps per task.



(e) Total scheduler workload (y-axis on log scale).



(f) Avg. task waiting time (y-axis on log scale).

(B) Total Number of Nodes=128.

Figure 6: The *SAP* Scheduling Strategy with a Fixed Set of Configurations. (A) Total nodes=64 and (B) Total nodes=128.

of Supercomputer Applications, vol. 11, pp. 115–128, 1996.

[3] S. J. Chapin, et. al, “The Legion Resource Management System,” in *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 162–178, 1999.

[4] D. Erwin, “UNICORE - A Grid Computing Environment,” in *Lecture Notes in Computer Science*, pp. 825–834, 2001.

[5] K. Compton and S. Hauck, “Reconfigurable Computing: A Survey of Systems and Software,” *ACM Computer Survey*, vol. 34, no. 2, pp. 171–210, 2002.

[6] TeraGrid, “FPGA Resources in Purdue University.” http://www.rcac.purdue.edu/teragrid/userinfo/fpga.

[7] S. Wong and M. Ahmadi, “Reconfigurable Architectures in Collaborative Grid Computing: An Approach,” in *Proceedings of the 2nd International Conference on Networks for Grid Applications (GridNets)*, 2008.

[8] F. Dong and S. G. Akl, “Scheduling Algorithms for Grid Computing: State of the Art and Open Problems,” *Technical report no. 2006-504*, 2006.

[9] H. Topcuoglu, S. Hariri, and M.-Y. Wu, “Task Scheduling Algorithms for Heterogeneous Processors,” in *Proceedings of the 8th Heterogeneous Computing Workshop (HCW)*, pp. 3–17, 1999.

[10] H. Casanova, “Simgrid: a Toolkit for the Simulation of Application Scheduling,” in *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid)*, pp. 430–437, 2001.

[11] H. J. Song, et. al, “The MicroGrid: a Scientific Tool for Modeling Computational Grids,” *Journal of Scientific Programming*, vol. 8, no. 3, pp. 127–141, 2000.

[12] R. Buya and M. M. Murshed, “GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing,” *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1175–1220, 2002.

[13] S. Wong, T. V. As, and G. Brown, “p-VEX: A Reconfigurable and Extensible Softcore VLIW Processor,” in *IEEE International Conference on Field-Programmable Technology (ICFPT)*, 2008.

[14] G. Marsaglia and W. W. Tsang, “The Ziggurat Method for Generating Random Variables,” *Journal of Statistical Software*, vol. 5, no. 8, pp. 1–7, 2000.

[15] G. Marsaglia and W. W. Tsang, “A Simple Method for Generating Gamma Variables,” *ACM Transactions on Mathematical Software*, vol. 26, no. 3, pp. 363–372, 2000.