Performance Analysis of Greedy Load Balancing Algorithms in Heterogeneous Distributed Computing System

Bibhudatta Sahoo
Department of Computer Science and Engineering
NIT Rourkela, Odisha, India
Email: bdsahu@nitrkl.ac.in

Dilip Kumar
Department of Computer Science and Engineering
NIT Jamshedpur, India
Email: dilip.cse@nitjsr.ac.in

Sanjay Kumar Jena
Department of Computer Science and Engineering
NIT Rourkela, Odisha, India
Email: skjena@nitrkl.ac.in

***Abstract*—The Load balancing problem on Heterogeneous Distributed Computing System (HDCS) deals with allocation of tasks to compute nodes, so that computing nodes are evenly loaded. Dynamic load balancing problem to assign tasks on HDCS is presented as a linear programming problem to minimize the *makespan*. Different greedy resource allocation algorithms are presented for load balancing on HDCS with tasks as Expected Time to Compute(ETC) matrix. The relative performance of the four heuristic algorithms under different circumstances has been simulated on two different HDCS using in house simulator. The simulation results show that the greedy based scheduling policy has the significant impact on the system heterogeneity.**

***Keywords*—Expected time to compute, greedy algorithm, heterogeneous distributed system, load balancing, makespan.**

I. Introduction

Heterogeneous Distributed Computing platforms are widely used to process various jobs from different field of scientific applications. The potential of distributed computing system is relates to the management and allocation of computing resources relative to the computational load of the system [1][2][3][4]. The popular distributed computing platform includes clusters, the grid, service-oriented architecture, massively parallel processors, pear-to-pear networking, and cloud computing. These computational environments are consisting of multiple heterogeneous computing modules, these modules interact with each other to solve the problem. Balancing the computing loads among the computing nodes in a HDCS are carried out by the central server that assigns the jobs to the nodes, so as to optimize the makespan. Load balancing has been studied by various researchers as a problem to minimize the makespan [2][3][5][6]. The different techniques and methodologies for scheduling processes of a distributed system are task assignment, load-balancing, or load-sharing [1]. In HDCS, jobs encounter different execution times on different processors. Therefore, research should address scheduling in heterogeneous environment as the *load balancing* problem, that computes the assigned task with the smallest possible makespan. In this paper the load balancing problem is presented as minimization problem, to minimize the makespan of n tasks on m computing nodes [2][7]. The problem of finding an assignment of minimum makespan is NP-hard [7]. The algorithm approaches used for load balancing problem are roughly classified as (i) exact algorithms,(ii) heuristic algorithms,and (iii) approximation algorithm [8]. The solution to dynamic load balancing problem in this paper has been discussed with heuristic approach under greedy algorithm paradigm.

The selection of load balancing algorithm mostly depends on the set of system parameters such as (i) system size, (ii) system load, (iii) system traffic intensity [1]. In dynamic resource allocation scenarios the responsibility for making global scheduling decisions are lying with one centralized scheduler, or be shared by multiple distributed schedulers [9]. Hence dynamic load balancing algorithms can be further classified into a centralized approach and a decentralized approach. Dandamudi [10] has presented a study on impact of heterogeneity and variance in inter-arrival time in HDCS with two different arrival rate of the tasks to central scheduler. A centralized approach [2], [11] one node in the distributed system acts as the central controller and responsible for task allocation to other computing nodes. Casavant and Kuhl [12] have characterized the structure and behavior of decision-making policies, in particular referring to the load sharing policies considering performance and efficiency.

Dynamic load balancing algorithm based on task classification has been presented by Wang *et al.*[13]. Centralized static resource allocation algorithms has been studied extensively in [6], [14] using heuristics. Gopal $et\ al.$ [4] presented a simulation study for four load balancing algorithm on heterogeneous distributed system with the central job dispatcher. Izakian *et al.* [15] have presented an efficient heuristic method for scheduling independent tasks on heterogeneous distributed environment and compare with five popular heuristics for minimizing the *makespan*. In HDCS the load balancing is a job scheduling policy which takes a job as a whole and assign it to the computing node [1]. The rest of the paper is organized as follows. *Section 2* defines the system model of Heterogeneous distributed computing system model, task model and the load-balancing problem. *Section 3* presents the different greedy algorithms and their applicability to solve dynamic load balancing problem. Simulation results are presented in *Section 4*. Conclusions and directions for future research are discussed in *Section 5*.

II. System model for dynamic load distribution

A. Centralized system model for HDCS

The scheduling problem in HDCS aims to maintain a balanced execution of tasks while using the computational resources with computing node. Dynamic resource allocation in HDCS can be possible through centralized or decentralized control. A centralized dynamic load balancing algorithm operates based on the load information from another computing node and can realize through a centrally controlled HDCS. A centralized model of HDCS consists of a set of $M = \{M_1, M_2, ..., M_m\}$, m independent heterogeneous, uniquely addressable computing nodes, with one node acts as the resource manager. The single computing node that acts as a central scheduler or resource manager of the system is responsible for collecting the global load information of other computing nodes and allocate the task among the computing nodes. The centralized HDCS can be modeled as, $M/M/m$ (Markovian arrivals, Markovian distributed service times, m computing nodes as a server, and with infinite buffer for incoming task) multi-server queuing system. Allocation of a task to the computing nodes are equally probable and can be assigned by the central scheduler independently [16]. It is assumed that if all the computing nodes are busy the task will keep waiting in the waiting queue with central scheduler which is of infinite length.

B. Task or work load model

The work load submitted to the HDCS is assumed to be in the form of tasks. Depending on dynamic scheduling approach, the tasks are submitted either to the central scheduler or submitted to different computing nodes independently. For different domains of computer science the exact meaning varies greatly. Terms such as application, task, sub task, job and program is used to denote the same object in some instances, and yet, have totally different meanings in others. We have assumed the task as the computational unit to execute on the computing nodes of HDCS.

A task in HDCS is an independent scheduling entity and its execution cannot be preempted. The tasks are independent and can be executed on any node. Formally, each arriving task t_i is associate with an arrival time and expected time to compute on different computing node. Let T be the set of task, $T = \{t_1, t_2, ..., t_n\}$. Each task t_i, has an expected time to compute on node M_j, and dented as t_{ij}. Considering heterogeneity of system, tasks are characterized by Expected Time to Compute(ETC) matrix, where all m computing nodes can be represented in the first row. In ETC matrix, the elements along a row indicate the execution time of a given task on different nodes [17]; in particular t_{ij} represent expected time to compute i^{th} task on machine M_j. If the HDCS has m computing nodes, then they can be represented by ETC matrix. Hence a program with n task can be represented as a $n \times m$ ETC matrix on m computing node. Hence ETC matrix can be used to study dynamic load balancing problem in HDCS. Because there are no dependency among the tasks, load balancing schemes is simplified, and mostly focuses on efficient matching of tasks to the computing nodes [14]. It is assumed that the size of the *meta-task* is the number of tasks to be executed on HDCS and denoted as $|T| = n$.

The ETC model presented in [17] are with three parameters (i) machine heterogeneity, (ii) task heterogeneity and (iii) consistency. The task heterogeneity can be represented with two categories (i) consistent and (ii) inconsistent. A *consistent ETC matrix* can be obtained by arranging the computing nodes in order of their processing capability or may be arranged as decreasing order of FLOPS. In particular a node M_i has a lower execution time than node M_j for task t_k , then $t_{ki} < t_{kj}$. The *Inconsistent ETC matrix* is resulted in practice, when HDCS includes different type of machine architectures such as high performance computing clusters, multi-core processor based workstations, parallel computers, work station with GPU units. In literature, most of the researchers used the $task\ execution\ times$ as uniformly distributed [17]. In this paper, we have used *Inconsistent ETC matrix* to study the performance of Greedy Load Balancing algorithms in HDCS.

C. Dynamic Load balancing as linear programming problem(LPP)

Dynamic load balancing problem to assign n tasks on HDCS with m computing nodes can be represented as a optimization problem to minimize the *makespan*. The task to be executed on HDCS are represented by the ETC matrix [17]. Let $A(j)$ be the set of task assigned to node M_j; and T_j be the total time machine M_j have to work to finish all the task in $A(j)$. Hence $T_j = \sum_{t_i \in A(j)} t_{ij}$; for all task in $A(j)$. This is otherwise denoted as L_j and defined as load on node M_j. The basic objective of load balancing is to minimize the *makespan*, which is defined as maximum loads on any node ($T = max_{j:1:m} T_j$). Let x_{ij} correspond to each pair (i, j) of node $M_j \in M$ and task $t_i \in T$ such that

$$x_{ij} = 0;\ when\ the\ task\ i\ not\ assign\ to\ node\ M_j. \quad (1)$$

or

$$x_{ij} = t_{ij};\ when\ the\ load\ of\ task\ i\ on\ node\ M_j. \quad (2)$$

For each task t_i, we need $\sum_{j=1}^{m} x_{ij} = t_{ij}$;for all task $t_i \in T$. The load on node M_j can be represented as $L_j = \sum_{j=1}^{m} x_{ij}$, where x_{ij} is defined in Equations 1 and 2. The load balancing problem aims to find an assignment that minimizes the maximum load. Let L be the load of HDCS with m nodes. Hence the generalized load balancing problem on HDCS can be formulated as

$$Minimize\ L = \sum_{j=1}^{m} x_{ij} = t_{ij},\ \forall\ t_i \in T \quad (3)$$

subjected to:

$$\sum_{j=1}^{n} x_{ij} \leq L,\ \forall\ M_j \in M \quad (4)$$

$$where\ x_{ij} \in \{0, t_{ij}\}, \forall t_i\ \in T, and\ M_j \in M$$

$$x_{ij} = 0,\ \forall\ t_i \notin A(j)$$

The objective Function 3 maps each possible solution of the load balancing problem to some non-negative value, and an optimal solution to the optimization problem is one that minimizes the value of this objective function. Feasible assignments are one-to-one correspondence with x_{ij} satisfying the constraints in Equation 4. Hence an optimal solution to this problem is the load L_j on a node, also denoted as corresponding assignment $A(j)$. For n tasks to be assigned to m computing node, the number of possible allocation will be m^n and the number of states for execution will be $n!$, hence intractable with number tasks or computing nodes exceeds a few units [7].

III. Greedy heuristic algorithms for load balancing

A. Greedy load balancing algorithm

All heuristic algorithms for load balancing discussed in this paper follows the simple *Greedy-Balance* algorithm framework, discussed by Kleinberg and Tardos [7] to suggest a generalized greedy load balancing algorithm for HDCS. The proposed algorithms operates with the *ETC matrix* and *arrival time* for each task, to allocate the task to computing node for load balancing. The arrival time of the task to be recorded in a priority queue $HAT(MaxTask)$. The priority queue $HAT(MaxTask)$ has been implemented as *min-heap*, records the order at which the tasks are arriving at central scheduler. The *min-heap* can be created with time complexity $\bigcirc(log\ n)$. The task with earliest arrival time selected and assigned to the machine with minimum load. The assumption made to design the greedy algorithm is that *"initial load of computing nodes are zero"*. The greedy Algorithm 1 is designed to obtain an optimal task assignment by assigning the n tasks in stages, one task per stage in non-decreasing order of task arrival time. The greedy load balancing algorithm operates by initializing set of task $A(j)$ and the total time to finish the task T_j for every node M_j. The Algorithm 1 can be implemented with time complexity $\bigcirc(n\ log\ n)$. The Algorithm 1 successfully terminates with task queue becomes empty. The selection of computing node is based upon greedy criterion : *assign the task to the node with minimum* T_j. The algorithm computes *makespan* for the set of task having $MaxTask$ number of tasks.

Algorithm 1 Greedy load balancing algorithm with priority queue

```
Require: $ETC(MaxTask, MaxNode), HAT(MaxTask) : task\ Queue$
Ensure: $L : makespan$
 1: $L_j \longleftarrow 0\ forall$ node $M_j$
 2: $A(j) \longleftarrow \phi\ forall$ node $M_j$, Let $\phi$ be the empty set
 3: repeat
 4:     Let $M_j$ be a node with minimum $T_j$
 5:     Let $t_i$ be the task on root of the min-heap $HAT$
 6:     Allocate task $t_i$ to Node $M_j$
 7:     $A(j) \longleftarrow A(j) \cup \{t_i\}$
 8:     $L_j \longleftarrow L_j + t_{ij}$
 9:     Remove task $t_i$ from min-heap $HAT$
10: until $HAT$ is not empty
11: $L \longleftarrow max_j L_j$
```

Objective of dynamic load balancing algorithm to allocate the tasks *on the fly* as the tasks are queued with the central scheduler with Poission arrival. A fixed batch size denoted as $WinSize$ and defined as the number of tasks selected from a batch for allocation. As there are too many tasks waiting with central scheduler to be allocated, the scheduling heuristics only apply to the tasks that are within the batch. Once one batch of tasks are allocated to the computing nodes, the next batch of task is selected from the task queue for allocation. The Algorithm 1 can modify to Algorithm 2 with $MaxTask = WinSize$ to facilitate batch mode resource allocation. The Algorithm 2 can be called $\frac{n}{WinSize}$ times in batch mode to allocate n tasks dynamically to m computing node. For simplicity it is assumed that $MaxTask$, is positive integer multiple of $WinSize$. The Algorithm 2 is initiated first time with following initiation:

$$L_j \longleftarrow 0\ for\ all\ node\ M_j$$

$$A(j) \longleftarrow \phi\ for\ all\ node\ M_j$$

The *makespan* can be obtained after $\frac{n}{WinSize}$ steps as $L = max_j L_j$. Heuristic and meta-heuristic algorithms are

Algorithm 2 Greedy load balancing for batch job with priority queue

```
Require: $T, A, ETC(WinSize, MaxNode), HAT(WinSize) : task\ Queue$
1: repeat
2:     Let $M_j$ be a node with minimum $L_j$
3:     Let $t_i$ be the task on root of the min-heap $HAT$
4:     Allocate task $t_i$ to Node $M_j$
5:     $A(j) \longleftarrow A(j) \cup \{t_i\}$
6:     $L_j \longleftarrow L_j + t_{ij}$
7:     Remove task $t_i$ from min-heap $HAT$
8: until $HAT$ is not empty
```

the effective technology for scheduling in HDCS due to their ability to deliver high quality solutions in reasonable time. The dynamic load balancing algorithm using batch mode heuristic MINMIN and MINMAX operates by selecting fixed small number that fits to the task window on each iteration. The MINMIN and MINMAX operates for the fixed number of iteration to assign n tasks to the computing nodes.

B. First-Come, First-Served (FCFS) heuristic

The FCFS heuristic is a very simple and most common resource allocation heuristic are being used by various researchers to study task scheduling in distributed system[2], [10], [14]. This is a non-preemptive scheduling policy that schedules tasks in the order of their arrival to the central scheduler. The FCFS Algorithm 3 is applied to the load balancing problem discussed in Section II-C. A *min-heap* is created to maintain the order of the task as per their *time of arrival* to the system and represented as HAT. The load status of the computing node M_j is represented as CL_j. Every iteration assigns the task with least arrival time to a computing node M_j with $CL_j = Null$ in HDCS.

C. Randomized algorithm

The *randomized algorithm* used in this paper is a Monte Carlo algorithm as it runs for a fixed number of steps equal to the maximum number of tasks to be assigned. The result

Algorithm 3 FCFS

```
Require: $T\ :\ set\ of\ task, M\ :\ set\ of\ node, ETC\ :\ expected\ time\ to\ compute, HAT : task\ Queue$
Ensure: $A : Allocation\ List, L : makespan$
 1: $L_j \longleftarrow 0\, for\ all$ node $M_j$
 2: $A(j) \longleftarrow \phi\, for\ all$ node $M_j$
 3: repeat
 4:   let $t_i$ is the task at root of min-heap $HAT$
 5:   $allocate \longleftarrow\ false$
 6:   repeat
 7:     for $j = 1$ to MaxNode do
 8:       if $CL_j$ = Null then
 9:         Allocate task $t_i$ to Node $M_j$
10:         Remove task $t_i$ from min-heap $HAT$
11:         $A(j) \longleftarrow\ A(j) \cup \{t_i\}$
12:         $L_j \longleftarrow\ L_j + t_{ij}$
13:         $allocate \longleftarrow\ true$
14:       end if
15:     end for
16:   until allocate = false
17: until $HAT$ is not empty
18: $L \longleftarrow max_j L_j$
```

produced by randomized algorithm are not optimal, but characterized by some probability to represent the average case, hence are used to compare the performance of another deterministic algorithms. The details of the randomized resource allocation algorithm are shown in Algorithm 4. Each iteration select the task from the root of min-heap HAT and allocates to a randomly selected computing node. The time complexity of Algorithm 4 is an $\bigcirc(n)$ to assign n tasks to m computing node.

Algorithm 4 Random

```
Require: $T\ :\ set\ of\ task, M\ :\ set\ of\ node, ETC\ :\ expected\ time\ to\ compute, HAT : task\ queue$
Ensure: $A : Allocation\ List, L : makespan$
 1: $L_j \longleftarrow 0\, for\ all$ node $M_j$
 2: $A(j) \longleftarrow \phi\, for\ all$ node $M_j$
 3: repeat
 4:   let $t_i$ is the task at root of min-heap $HAT$
 5:   let $M_j$ be a node selected at random
 6:   allocate task $t_i$ to Node $M_j$
 7:   $A(j) \longleftarrow\ A(j) \cup \{t_i\}$
 8:   $L_j \longleftarrow\ L_j + t_{ij}$
 9:   remove task $t_i$ from min-heap $HAT$
10: until $HAT$ is not empty
11: $L \longleftarrow max_j L_j$
```

D. MINMIN algorithm

The MINMIN algorithm is a dynamic task allocation algorithm on HDCS operates in batch mode, and realized through discrete event simulation. Min-Min heuristics use the ETC matrix to compute completion time for n number of tasks. Algorithm 5 represents heuristic algorithm for HDCS and named as MINMIN. This algorithm considers all the unmapped tasks during each allocation decision but maps only one task at a time. Every allocation of task to the computing node is followed by update of expected completion time of all unallocated tasks. Let the task t_k is having minimum expected completion time on node M_l, *i.e.* $C_{kl} = min(C_{k1}, C_{k2}, \cdots, C_{km})$. Algorithm 5 allocates the task t_k to the computing node M_l as the task t_k is having minimum expected completion time with node M_l. The *makespan* is computed after the complete allocation of all tasks as $L = max_j L_j$.

Algorithm 5 MINMIN

```
Require: $T\ :\ set\ of\ task, M\ :\ set\ of\ node, ETC\ :\ expected\ time\ to\ compute$
Ensure: $A : Allocation\ List, L : makespan$
 1: for all task $t_i$ in meta-task $T$ do
 2:   for all machine $M_j$ in $M$ do
 3:     $C_{ij}\ \longleftarrow\ t_{ij}\ +\ L_j$
 4:   end for
 5: end for
 6: repeat
 7:   for all task $t_i$ in $T$ do
 8:     find the task with minimum completion time. Let $t_k$
        be the task with minimum completion time on node
        $M_l$
 9:   end for
10:   assign task $t_k$ to node $M_l$
11:   update load of node $M_l$ as $L_l \longleftarrow\ L_l + t_{kl}$
12:   update $C_{il}$ for all unallocated task
13:   Remove task $t_k$ from task list $T$
14: until $T$ is not empty
15: $L \longleftarrow max_j L_j$
```

E. MINMAX algorithm

This algorithm is different from common Max-min algorithm defined in [14]. The Algorithm 6 is composed of two steps. The algorithm operates on the batch of task and their respective ETC matrix. The algorithm computes the completion time for all the tasks on HDCS. The first step is the selection of task with minimum completion time in HDCS with m node, Let t_k be the task with minimum completion time on node M_l. The first step is the same as MINMIN algorithm. The second step decides the allocation of task to a computing node, which can be decided as follows:

- If $\dfrac{t_{kf}}{t_{kl}}\ \geq\ C_{kl}$ then allocate the task t_k to node M_f
- else assign the task t_k to node M_l.

IV. Results and discussion

We have conducted extensive simulation with the in house simulator designed by us using Matlab, that uses $M/M/m$ queuing model to simulate the task arrival with heterogeneous computing nodes. The tasks are arriving with a rate λ to the central server queue. We consider only 500 task for these experiments, that uses inconsistent task model as suggested in [17] on a HDCS with 60 nodes. We have consider two types of heterogeneous system, where the HDCS are characterized by the service rate of the computing nodes. The two types of system used to study the impact of heterogeneity are: ***Type I***: The first system model includes half of the computing

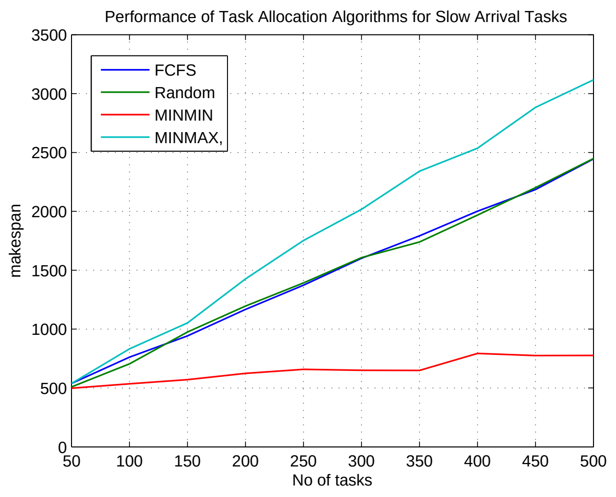

Fig. 1. Makespan with number of tasks in type-I system with slow arrival

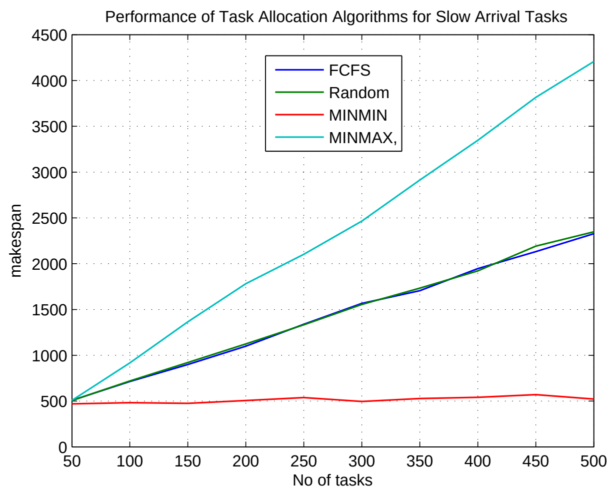

Fig. 4. Makespan with number of tasks in type-II system with slow arrival

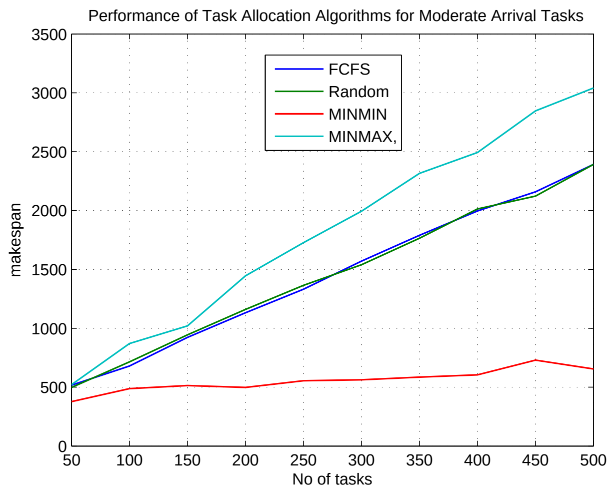

Fig. 2. Makespan with number of tasks in type-I system with medium arrival

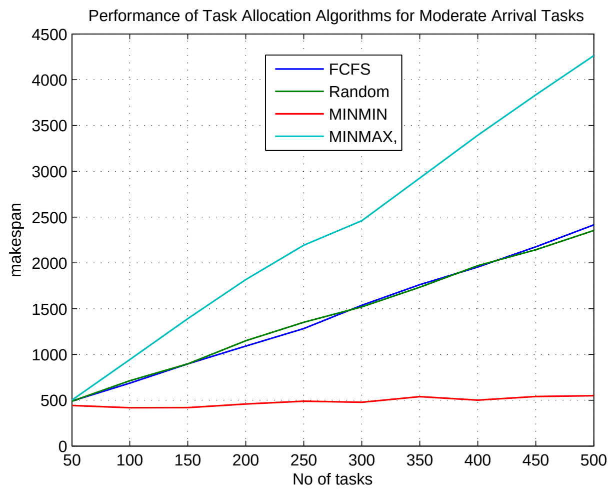

Fig. 5. Makespan with number of tasks in type-II system with medium arrival

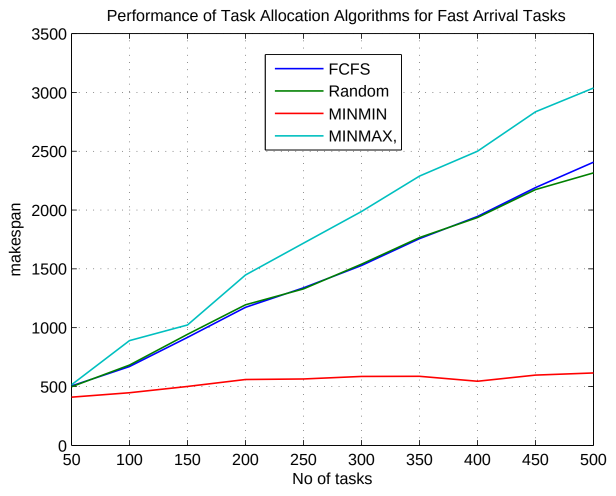

Fig. 3. Makespan with number of tasks in type-I system with fast arrival

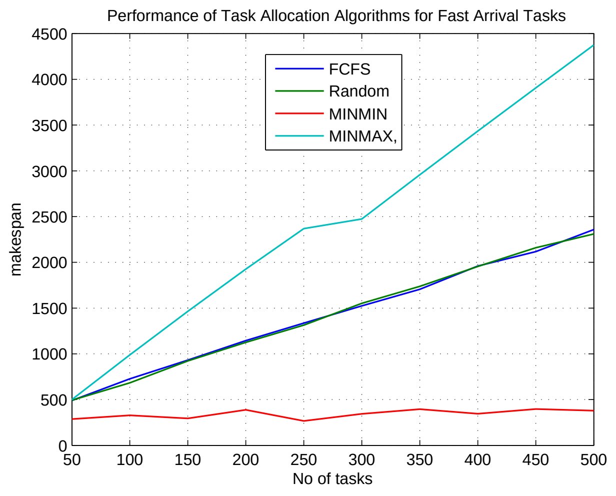

Fig. 6. Makespan with number of tasks in type-II system with fast arrival

Algorithm 6 MINMAX

```
Require: $T$ : $set\ of\ task, M$ : $set\ of\ node, ETC$ : $expected\ time\ to\ compute, HTA : taskqueue$
Ensure: $A : Allocation\ List, L : makespan$
 1: for all task $t_i$ in meta-task $T$ do
 2:     for all machine $M_j$ in $M$ do
 3:         $C_{ij} \longleftarrow t_{ij} + L_j$
 4:     end for
 5: end for
 6: for all task $t_i$ in $T$ do
 7:     find the task with minimum completion time, Let $t_k$ be the task with minimum completion time on node $M_l$
 8: end for
 9: repeat
10:     if $\frac{t_{kf}}{t_{kl}} \geq C_{kl}$ then
11:         assign task $t_k$ to node $M_f$
12:         update load of node $M_f$
13:         update $C_{if}$ for all i
14:     else
15:         assign task $t_k$ to node $M_l$
16:         update load of node $M_l$
17:         update $C_{il}$ for all i
18:     end if
19:     Remove task $t_i$ from task list $T$
20: until $T$ is not empty
21: $L \longleftarrow max_j L_j$
```

nodes are homogeneous and other half of the computing nodes are heterogeneous with different service rate, and ***Type II***: The second system model includes the computing nodes with different service rate. The system was evaluated with slow, medium and fast load. The arrival rate of tasks are assumed to be 10, 20 and 30 for slow, medium, and fast arrival respectively. We have used the inconsistent ETC matrix to study the impact of heterogeneity in HDCS [17]. The impact on the inconsistent tasks arrived rate with Poisson distribution are shown in Figures 1, 2 and 3. The MINMIN heuristic performs better with more number of tasks in the system at a particular time instant. The results obtained also indicates that *FCFS* and *Randomized* algorithms exhibit similar performance on allocating the tasks using inconsistent task model. Simulation results on the HDCS with heterogeneous computing nodes are shown in Figures 4, 5 and 6. These simulation results clearly indicates the better performance of MINMIN algorithm. Moreover in three alternatives both FCFS and Randomized algorithms exhibit similar performance in terms of *makespan*. It also indicates highest *makespan*, average *makespan* and minimum *makespan* value against varying task.

V. Conclusion

The dynamic load balancing problem is modeled as an minimization problem. Load balancing is being performed during runtime at various stages to keep the workload balance on computing nodes of a HDCS. Simulation experiment were conducted to examine the performance of greedy resource allocation algorithms against *makespan* to study the task and node heterogeneity in HDCS by considering three different arrival rate for the tasks. It is observed that *batch mode* heuristics makes better decision, because these heuristics have the resource requirement information for the meta-task. It is also observed that the performance of greedy scheduling algorithm is also affected by the rate of heterogeneity of the tasks and number of computing nodes.

References

[1] J. Wu, *Distributed System Design.* CRC press, 1999.

[2] A. Zomaya and Y. Teh, "Observations on using genetic algorithms for dynamic load-balancing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 12, no. 9, pp. 899–911, 2001.

[3] H. Siegel, H. Dietz, and J. Antonio, "Software support for heterogeneous computing," *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 237–239, 1996.

[4] T. Gopal, N. Nataraj, C. Ramamurthy, and V. Sankaranarayanan, "Load balancing in heterogenous distributed systems," *Microelectronics Reliability*, vol. 36, no. 9, pp. 1279–1286, 1996.

[5] S. Dhakal, B. Paskaleva, M. Hayat, E. Schamiloglu, and C. Abdallah, "Dynamical discrete-time load balancing in distributed systems in the presence of time delays," in *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 5. IEEE, 2003, pp. 5128–5134.

[6] T. Braun, H. Siegel, A. Maciejewski, and Y. Hong, "Static resource allocation for heterogeneous computing environments with tasks having dependencies, priorities, deadlines, and multiple versions," *Journal of Parallel and Distributed Computing*, vol. 68, no. 11, pp. 1504–1516, 2008.

[7] J. Kleinberg and E. Tardos, *Algorithm Design.* Pearson Education Inc., 2006.

[8] G. Attiya and Y. Hamam, "Two phase algorithm for load balancing in heterogeneous distributed systems," in *Parallel, Distributed and Network-Based Processing, 2004. Proceedings. 12th Euromicro Conference on.* IEEE, 2004, pp. 434–439.

[9] F. Dong and S. G. Akl, "Scheduling algorithms for grid computing: State of the art and open problems," *School of Computing, Queens University, Kingston, Ontario*, 2006.

[10] S. Dandamudi, "The effect of scheduling discipline on dynamic load sharing in heterogeneous distributed systems," in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1997. MASCOTS'97., Proceedings Fifth International Symposium on.* IEEE, 1997, pp. 17–24.

[11] H. Lin and C. Raghavendra, "A dynamic load-balancing policy with a central job dispatcher (LBC)," *Software Engineering, IEEE Transactions on*, vol. 18, no. 2, pp. 148–158, 1992.

[12] T. Casavant and J. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *Software Engineering, IEEE Transactions on*, vol. 14, no. 2, pp. 141–154, 1988.

[13] H.-b. Wang, Z.-y. Fang, G.-n. Qu, and X.-d. Ren, "An innovate dynamic load balancing algorithm based on task classification," *IJACT: International Journal of Advancements in Computing Technology*, vol. 4, no. 6, pp. 244–254, 2012.

[14] M. D. Theys, T. D. Braun, H. J. Siegal, A. A. Maciejewski, and Y. Kwok, "Mapping tasks onto distributed heterogeneous computing systems using a genetic algorithm approach," *Solutions to Parallel and Distributed Computing Problems: Lessons from Biological Sciences*, pp. 135–178, 2001.

[15] H. Izakian, A. Abraham, and V. Snasel, "Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments," in *Computational Sciences and Optimization, 2009. CSO 2009. International Joint Conference on*, vol. 1. IEEE, 2009, pp. 8–12.

[16] B. Sahoo, S. Mohapatra, and S. K. Jena, "A genetic algorithm based dynamic load balancing scheme for heterogeneous distributed systems," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2008, Las Vegas, Nevada, USA, July 14-17, 2008, 2 Volumes*, 2008, pp. 499–505.

[17] S. Ali, H. J. Siegel, M. Maheswaran, and D. Hensgen, "Task execution time modeling for heterogeneous computing systems," in *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th.* IEEE, 2000, pp. 185–199.