HIGH THROUGHPUT COMPUTING DUE TO NEAR-OPTIMAL EMERGENT MULTIAGENT COALITIONS FOR LOAD SHARING

Leland Hovey and Mina Jung
Syracuse University, Syracuse, U.S.A.

Keywords: Multiagent, Evolution, Load-sharing, Scheduling.

Abstract: Grid CPU load-sharing is a subclass of computational grid resource management. Its purpose is to improve grid throughput – High Throughput Computing (HTC). The problem is load-sharing optimization state-space can be quite large. This is because of two factors: the load-sharing optimization problem is NP-complete, and a large volume of CPU-intensive loads can require thousands of Internet connected CPUs. Approximate models can find near-optimal solutions to NP-complete problems. Multiagent coalition formation (MCF) is a particular approximate game theoretic approach for these problems. We propose a new distributed MCF (DMCF) model for Grid CPU load-sharing, DMCF grouping genetic algorithm (DMCF-GGA). This paper presents the model in detail. It also compares this model with our existing model, DMCF-spatial. The comparison consists of a discussion of the models' similarities and differences, and a comprehensive empirical evalution. The results of this study are the following: The optimization search cost of DMCF-GGA is significantly less than DMCF-spatial. DMCF-GGA has a linear relation between coalition size and search cost (for high throughput). We have found preliminary lower and upper bound estimates for the effective coalition size. We have also found the average job sizes required for the run time of DMCF-GGA to be 1% of the job execution time.

1 INTRODUCTION

Currently, computing power for large-scale problem solving is in demand(Foster and (Eds.), 1999). If this power is provided by by a vast collection of small workstations (grid) instead of a single supercomputer, the financial cost is much less. Grid computing has evolved to be defined as "flexible, secure, coordinated resource management among dynamic multi-institutions", conjoined through the Internet or a dedicated network. Resource management is an optimized and dynamic assignment of distributed heterogeneous Grid resources. Optimization metrics include throughput, turnaround time, utilization, monetary cost, or access rights (Ibaraki and Katoh, 1988).

Recent examples of Grids, such as the Large Hadron Collider (LHC) and Fermi Lab experiments, have demonstrated the importance of resource management (RM) and have drawn much active grid RM research a RM optimized for certain metrics can provide the capacity for the large-scale job quantities produced by these experiments. The Open Grid Forum (OGF) and the Globus Alliance are also major organizations committed to grid research. Both organizations have specific RM research groups.

A computational grid is a consortium of distributed CPUs inter-connected by the Internet or dedicated links. The purpose is high throughput for large quantities of CPU-intensive jobs (such as found at LHC and Fermi). CPU load sharing is a subclass of RM. Optimized load sharing improves computational grid throughput by assigning loads (scheduling) so the load level of all CPUs is close to their capacity. But, since this problem is NP-complete (Fiala and Paulusma, 2005), and a Grid can have thousands of inter-connected CPUs, the load-sharing optimization state-space can be huge.

Approximation models can solve certain optimization problems having large state-spaces (Vazirani, 2004). Multiagent coalition formation (Sandholm, 1999) (MCF) is a type of approximate game theoretic model. MCF enables self-interested agents to reduce state-space search costs by coordinating their activities with other agents (e.g., the coordination of load-sharing among collections of CPUs). This paper proposes a new distributed MCF (DMCF) model for Grid CPU load-sharing, DMCF grouping genetic algorithm (Michalewicz, 1999) (DMCF-GGA). The motivation for this model is it is pragmatic in terms of algorithm complexity, low cost in terms of

Hovey L. and Jung M..
HIGH THROUGHPUT COMPUTING DUE TO NEAR-OPTIMAL EMERGENT MULTIAGENT COALITIONS FOR LOAD SHARING.
DOI: 10.5220/0003733702950305
In *Proceedings of the 4th International Conference on Agents and Artificial Intelligence* (ICAART-2012), pages 295-305
ISBN: 978-989-8425-95-9

both searching and communication, and scalable. The study explains this model is described in detail and presents the algorithm. It also compares this model with our existing model, DMCF-spatial. This consists of an explanation of the models' similarities and differences, and their empirical evaluation.

1.1 Objectives and Organization

The objectives of this study are a problem state-space reduction, and a cost/benefit analysis of whether autonomous agents can acquire the Core (section 2.2) for high throughput coalitions at the smallest search cost. The remainder of this paper is organized as follows: the background information and details are provided in Section 2. Section 3 describes the new model and explains the similarities and differences of the this model and the existing model. Section 4 describes all experiments utilized for the models' comparison. Finally, section 5 discusses implications and future directions.

2 BACKGROUND

This section consists of the following: Section 2.1 explains how the CPU load sharing *problem* fits within the field of Scheduling Theory (ST). It also shows the classification of our models within all types of ST *solutions*. Section 2.2 is a detailed description of the DMCF approach.

2.1 Problem and Solution Classifications

Generally, most problems within ST have the attributes listed by fig. 1 (Brucker, 2004). This paper's problem, CPU load sharing, is a specific scheduling problem characterized by the underlined items of the figure. This problem will serve as a basis for future work encompassing other attributes. The problem is also known as the generalized assignment or multiple knapsack problem.

Most scheduling solutions within ST have the attributes provided by fig. 2. Both DMCF solution models discussed in this paper are characterized by the underlined items of the figure. The first set of solution attributes is deterministic vs. non-deterministic. Efficient deterministic solutions to NP-complete problem have not been found. But, approximate non-deterministic solutions are an active research area. Both the compared models are a hybrid game theoretic/evolutionary algorithm. They are game theoretic since they attempt to attain stability in the Core through DMCF (section 2.2). They are also evolutionary because the search for a coalition structures is based on an evolutionary algorithm. The second set of attributes is static vs dynamic. Both models are dynamic since nodes can be added or taken away without changing the algorithms. The third set of attributes is centralized or decentralized. For a centralized algorithm (Wu et al., 2004), a single machine collects load data and determines the optimal allocation. This locality of control can provide algorithm efficiency and easy management. But, these algorithms are not scalable and fault tolerant. The Hungarian Method (Kuhn, 1955) and "Mulknap" (Pisinger, 1999) are both existing solutions to the multiple knapsack problem. Since they are both centralized, the cost of large problem instances can be prohibitive. Decentralized algorithms (Csari et al., 2004; Weichhart et al., 2004) divide the overall assignment task among mutiple sites. These sites can act as both an allocator and a computing resource. Since no site per-

A. **Machines and Architecture**
   1. Heterogeneous vs. identical machines
   2. Multi-purpose vs. uni-purpose machines
   3. Static vs. dynamic machine availability
   4. Resource allocation vs. job scheduling
   5. Metascheduling vs. local scheduling

B. **Job Characteristics**
   - Multiprocessor vs uniprocessor jobs
   - Queue size vs. job size
   - Divisible vs. non-divisible jobs
   - Jobs with deadlines
   - Job size estimation
   - Job preemption and migration
   - Job arrival distribution

C. **Optimality Metrics**
   - Makespan, or turnaround time
   - Total flow time, total of turnaround times
   - Weighted (total) flow time
   - Total throughput
   - Earliness
   - Lateness
   - Square deviation

Figure 1: Grid Scheduling Problem Classification: (1) Machines and Architecture, (2) Job Characteristics, and (3) Optimality Metrics.

A. **Deterministic vs. non-deterministic**
   - Probabilistic
   - Control theoretic
   - Game theoretic
   - Genetic
   - Hierarchical

B. **Static vs. dynamic**

C. **Centralized vs. decentralized**

Figure 2: Scheduling Theory Solution Classification.

forms the entire assignment task, decentralized algorithms can be scalable and fault tolerant. However, these algorithms may incur high communication overhead (usage monitoring). Also, a centralized algorithm can be closer to optimal than multiple local allocators.

2.2 DMCF Overview

A characteristic function game (CFG) is a game in which a characteristic function determines the value of each coalition V_S, where S is a coalition. MCF is a type of CFG that consists of three phases:

I. Coalition structure generation: construct of partition[1] of agents where each subset of agents is a coalition. This partition is called a coalition structure (*CS*). Social welfare is the sum of all agent's payoffs. The goal of this phase is to maximize the social welfare of agents A by finding a coalition structure[2]

$$CS^* = arg_{CS \in \text{ partitions of } A} \max V(CS),$$
$$\text{where } V(CS) = \sum_{S \in CS} v_S.$$

Both of this study's models partition the overall n autonomous multiagent load sharing problem into k, ($k < n$) load sharing subproblems. There is a coalition of agents for each subproblem. Each subproblem consists of: (1) a coalition of multiagents where each agent (called a node-agent) acts on behalf of each CPU node, (2) each node-agent having the task of potentially sharing its load with some other coalition node.

II. Solve the optimization problem of each coalition. First, share the tasks and resources of the agents in the coalition. Next, calculate $\forall S \in CS, v_S$. Finally, solve the joint problem. In this study, DMCF-spatial uses a locality based technique and DMCF-GGA uses the first fit algorithm.

III. Payoff – dividing the value of the generated coalition structure among agents. The *Core* is a specific payoff scheme defined as the agents remaining within the coalitions instead of moving out of them:

$$Core = (\vec{x}, CS) \; \forall \, S \subset A, \; \sum_{i \in S} x_i \geq v_S$$
$$and \; \sum_{i \in A} x_i = \sum_{S \in CS} v_S.$$

A *CS* that maximizes social welfare is stable in the Core (Sandholm, 1999). Both models optimize throughput and divide this amount evenly among all agents (e.g., social welfare). Both models terminate if no agents choose to move out of the final *CS*.

3 MODELS: DMCF-GGA AND DMCF-SPATIAL

This study compares two evolutionary models for large-scale Grid CPU load sharing optimization: the new DMCF-GGA and the previous DMCF-spatial. This section consists of the following: Section 3.1 discusses the similarities and differences of the two models. Next, section 3.2 describes the DMCF-GGA algorithm in detail and provides the algorithm code.

3.1 Similarities and Differences

Both models are based on the three-phased DMCF approach explained in section 2.2. The similarities of the two models occurs in Phase I as explained in section 3.1.1. Section 3.1.2 discusses the differences that occur both in Phase I and II.

3.1.1 Similarities

Both models have the following attributes in common for Phase I:

- The agents and algorithms operate in a distributed environment.
- The agents and algorithms minimize the communication overhead.
- The emerged *CS* (partition) maximizes total throughput (near-optimal).
- The models are pragmatic. Our intention for the designs and implementations is so they can be readily deployed as an additional scheduler in Condor (an existing grid batch job scheduling system).
- Coalition formation is due to an evolutionary algorithm. Each coalition is comprized of node-agents. Initially, the autonomous node-agents form a *CS* of k coalitions. Each generation, node-agents self-organize into a new *CS*. The sequence of generations causes a sequence of *CS*s. Since

[1] A partition of a set X is a set of nonempty subsets of X such that every element x in X is in exactly one of these subsets.

[2] Superadditivity is when any pair of coalitions is best off by merging into one. Our model prevents superadditivity by having a defined coalition size for each experiment.

the coalitions' members may change, the coalitions are dynamic. These evolving *CS*s have monotonically increasing fitness. The fitness metric is total throughput.

- A distributed chromosome represents a *CS* (the entire state of the Grid). Each of the chromosome's genes specifies a node's coalition. A gene is implemented as a node-agent. There is one node-agent per node. A node-agent acts on behalf of its node. Hence, the node-agents (chromosome) are distributed. Each generation, some of the gene values are replaced. There is one chromosome for the evolutionary algorithm.
- There are two types of distributed agents;
  - *condition-action* node-agents – the are agents that self-organize (evolve) into coalitions as described above. To construct new coalitions each generation based on a condition, node-agents may perform the action of a *move*. Condition-action *moves* are implemented as conditional migration, and
  - cm-agents – these agents distribute coalition member node-agents with `coalition member` and `throughput` lists, and they distribute these lists to the other coalitions. There is one cm-agent for each coalition.
- The genetic operator is conditional migration (Kowalski and Sadri, 1996). This enables the node-agents to form new coalitions. The procedure is: (1) a node-agent starts by randomly chosing a new candidate coalition, (2) all node-agents effected by this possible membership determines if the change increases both the throughput of the coalition where the candidate is from and the throughput of the coalition where the candidate move to, and (3) if so, the node-agent joins the coalition. The result of all migrations during a generation is an increase fitness (throughput).
- The average job size is measure in Mflops[3],

Phase II: the node-agents within each emergent coalition collaborate to share the job loads (e.g., load sharing). The load sharing algorithms differ for the two models (section 3.1.2). The output of this phase is a map that assigns each coalition job to a specific coalition node[4].

Phase III: the payoff is divided evenly among the node-agents when the agents are stable in the Core (when no further throughput increases occur).

3.1.2 Differences

For Phase I, the two models differ about: (1) the gene structure and (2) procedure for conditional migration. A DMCF-spatial gene is a pair of cartesian coordinates. So, each node-agent is located at a point on a 2-dimensional logical grid (Oliphant, 1994). Node-agents that have the same spatial proximity[5] belong to the same coalition. A DMCF-GGA chromosome is the same as the former. But, the gene of its node-agent is a coalition ID. DMCF-GGA node-agents with the same coalition ID belong to the same coalition.

DMCF-GGA is an example of Cooperative Distributed Problem Solving (CDPS) (Decker et al., 1998). Fig. 3 presents an evaluation of the agents (both node-agents and cm-agents) as a CDPS system. The node-agents operate independently to enable the coalition to emerge. Since there is no central locality of control, the failure of any node does not hinder operation of the algorithm. This improves reliability and fault tolerance. If more nodes are added, more coalitions are constructed. For the number of nodes we tested, the number of coalitions does not affect scalability.

A. **Heterogeneity of the system, structural assumptions, domain and architectural assumptions**
   1. agent functionality is identical
   2. number of agents equals the number of servers
   3. the solution evolves

B. **Effective coordination to approach common goals**
   1. agents shares problem solving knowledge through a cm-agent
   2. representation and reasoning of agent goals is optimal system throughput
   3. agent goals interrelate: agents' throughputs are averaged for coalition throughput and optimal system throughput
   4. range of agent collaboration: critical timing constraints for the actions of the agents.

C. **Organization and control of the system**
   1. task decomposition: a task consists of partitioning an individual into coalitions.
   2. task allocation: each server has pre-allocated tasks. Tasks are reallocated to the most efficient server.
   3. results collection: the coalition-manager agents collects the results from the node-agents. Then, the coalition-manager agents all exchange their results.

Figure 3: A cooperative distributed problem solving framework to describe the presented DMCF model.

DMCF-spatial's conditional migration moves a

[3]"Floating point operations per second (FLOPS) has been the yardstick used by most High Performance Computing (HPC) efforts to rank their systems (Livny et al., 1997)."

[4]A single application of our model has executed on grids up to 50000 nodes.

[5]A node-agent defines a circle of radius `r` that specifies if the nodes belong to the same coalition. Node-agents located within the radius have the same spatial proximity.

node-agent to the proximity of another coalition if the throughput of each affected coalitions improves. Possible coalition overlap is an ancillary effect. DMCF-GGA's conditional migration changes a node-agent's coalition ID if the throughput of each affected coalitions improves. Coalitions do not overlap. This model's motivation is to reduce of the load sharing problem's search space (e.g., search cost) over the DMCF-spatial model. DMCF-spatial's conditional migrations may consist of many attempts at relocating nodes at locations outside the neighborhood of every coalition. But, with DMCF-GGA every conditional migration is an attempt to join one of a small number of coalitions.

The DMCF-spatial algorithm for Phase II is the following: the load a node-agent shares with another coalition node-agent is inversely proportional to the number of coalitions where the second node-agent is a member. (Hovey et al., 2003) contains the algorithm details. Phase II for DMCF-GGA is the first fit algorithm.

3.2 DMCF-GGA Algorithm

First, this section describes the two types of DMCF-GGA agents, a node-agent and a cm-agent (fig. 4). Then, it explains the DMCF-GGA algorithm in detail.

node ID	coalition ID	node load	coalition throughput	push/pull algorithms

(a) node-agent

coalition ID	node load node load ⋮ node load	coalition throughput	push/pull algorithms

(b) cm-agent

Figure 4: The composition of a node-agent and a cm-agent.

A node-agent consists of the following five components (fig. 4(a)): (1) node ID - unique node specifier (0 .. 500), (2) coalition ID - specifies the coalition where the node-agent is a member - the range of which is [0 .. nCoalitions], (3) node load; the total of the jobsize of all jobs at the node (Mflops), (4) coalition throughput; the total throughput (Mflops) of all coalition members, and (5) algorithms to push/pull components 1 – 4. In addition, a node-agent remains located at a specific node throughout the DMCF-GGA algorithm.

A cm-agent has the following four components (fig. 4(b)): (1) coalition ID; specifies the coalition where the node-agent is a member (0 .. nCoalitions), (2) a node load (Mflops) for each of the coalition members, (3) the throughput of each coalition (Mflops) in the entire set of coalitions, and (4) algorithms to push/pull components 1 – 3. The location of each cm-agent is defined by an elect algorithm (section 3.2.2). It may change at any generation during Phase I. But, it remains the same for Phase II.

The DMCF-GGA algorithm (algo. 1) is summarized as follows. The top-level is: (1) `DistributedInitializeSystem`, then (2) `DistributedEvolveCoalitions`. The result is a *CS* having the highest total throughput. This procedure's variables are *AD* is an administrative domain[6], *pc* is the previous generation having a change in throughput, and *g* is the current generation.

Algorithm 1: DMCF-GGA Top-level: DistributedInitializeSystem and DistributedEvolveCoalitions.

```
1: procedure DISTRIBUTEDINITIALIZESYSTEM
2:     Update-self-AD-cm-agent
3:     Update-every-AD-cm-agent
4:     When each AD receives "sufficient Mflops" from every AD
   cm-agent, the system is initialized.
5:     NotifyReady
6: end procedure
7: procedure DISTRIBUTEDEVOLVECOALITIONS
8:     repeat
9:         ConditionalMigration
10:        NotifyDoneMigration
11:        CoalitionSnapshot
12:        NotifyDoneGeneration
13:    until
14: end procedure
```

3.2.1 DistributedInitializeSystem

`DistributedInitializeSystem` can be subdivided into 3 steps (algo. 1 lines 2–4). These second level procedures are presented in algo. 2 and they each run once[7].

1. `UpdateLocalCMA`, algo. 1 step 2, is listed as algo. 2 lines 1–6. At the start of DMCF-GGA, each AD has a server for all its nodes. This is the initial cm-agent (*cma*). It is chosen at random. Jobs may start to arrive at each node at this time. The node-agent records the total job size as the jobs arrive. If the total Mflops of the jobs is within a threshold of the node capacity, the node-agent sends "`Mflops ok`" to that AD's *cma* and job arrival is cutoff. The procedure's variables are: *cma* is a cm-agent, t is the total Mflops of all the jobs on a node, th_t is the required total jobsize threshold, and cap the capacity of the node (Mflops).

[6] An AD is typically a single administrative authority managing a collection of servers and routers, and the interconnecting network(s).

[7] at approximately the same time

2. `UpdateEveryCMA` (algo. 1 step 3) is listed as algo. 2 lines 7–10. When each cma_s receives "`Mflops ok`", they each send "`Mflops ok`" to every other cma_r cm-agent. This procedure's variables are: n is a node-agent member of a coalition, cma_s is the sender cma, and cma_r is the receiver cma.
3. `NotifyReady` (algo. 1 step 4) is listed as alg. 2 lines 11 – 19. First, each cma_r cm-agent receives "`Mflops ok`" from every cma_s. Next, each cma_r sends "`Mflops ok`" to each of its member node-agents. Then, the node-agents send "`ready`" to their cma_s. Each cma_s sends "`ready`" to every other cma_r. Finally, each cma sends "`attempt-mig`" to each of its member node-agents. This procedure's variables are cma_r is all nodes and they are acting as receivers.

Algorithm 2: Second-level: InitializeSystem Components.

```
1: procedure UPDATELOCALCMA
2:     ∀cma ∀n ∈ cma,
3:     if abs (t - cap) < th_t then
4:         send ("Mflops ok", n, cma);
5:     end if
6: end procedure
7: procedure UPDATEEVERYCMA
8:     ∀cma ∀n ∈ cma, recv ("Mflops ok", n, cma);
9:     ∀cma_s ∀cma_r, send ("Mflops ok", cma_s, cma_r);
10: end procedure
11: procedure NOTIFYREADY
12:     ∀cma_s ∀cma_r, recv ("Mflops ok", cma_s, cma_r);
13:     ∀cma_r ∀n ∈ cma_r, send ("Mflops ok", cma_r', n);
14:     ∀n ∈ cma_s ∀cma_s, send ("ready", n, cma_s);
15:     ∀n ∈ cma_s ∀cma_s, recv ("ready", n, cma_s);
16:     ∀cma_s ∀cma_r, send ("ready", cma_s, cma_r);
17:     ∀cma_s ∀cma_r, recv ("ready", cma_s, cma_r);
18:     ∀cma_r ∀n ∈ cma_r, send ("attempt-mig", cma_r, n);
19: end procedure
```

3.2.2 DistributedEvolveCoalitions

DistributedEvolveCoalitions can be subdivided as listed in algo. 1 steps 6–15. Each node-agent runs[7] its own copy of this procedure. This procedure's variables are:

$cma \equiv$ coalition manager agent
$c \equiv$ temporary coalition ID
$p \equiv$ probability of migration
$nc \equiv$ total number of coalitions
$n(t) \equiv$ throughput
$c_1 \equiv$ coalition where node-agent is migrating from
$c_2 \equiv$ coalition where node-agent is migrating to
$nm \equiv$ number of coalition members
$cap \equiv$ node capacity
$th_c \equiv$ coalition threshold
$c\_ID \equiv$ coalition ID

An explanation of these steps follows. Algo. 1 step 7: Repeat steps 8 – 12 until there is no increase of total throughput for 100 generations[8].

[8]For grid size above 500, the fitness approaches the maximum more slowly. So, the number of generations for the stopping condition increases.

`ConditionalMigration` (algo. 1 step 8) is listed as algo. 3. When each node-agent receives either "`ready`" or "`attempt-mig`", it attempts to migrate with probability "p". Temporarily, the node-agent randomly chooses a new coalition ID. A migration occurs if the condition (algo. 3: steps 8 – 11) is met. Intuitively, a node migrates if; a) it is overloaded, its coalition is overloaded, and the coalition it moves to is underloaded, or b) if it underloaded, its coalition is underloaded, and the coalition it moves to is overloaded. If this condition is met, the temporary new coalition becomes fixed (for at least for one generation). Then, each node-agent, whether or not it migrates, sends "`done-mig`" to its cm-agent.

Algorithm 3: Second-level: Conditional Migration.

```
1: procedure CONDITIONALMIGRATION
2:     ∀cma ∀n ∈ cma,
3:     recv ("attempt mig", cma, n);
4:     if (rand() < p) then
5:         c = int (urand[0..nc])
6:         ε1 = (n(t)_c1 − (nm × cap));
7:         ε2 = (n(t)_c2 − (nm × cap));
8:         if
9:         ((n(t)_n > cap) && ε1 > th_c && ε2 < th_c) ||
10:        ((n(t)_n < cap) && ε1 < th_c && ε2 > th_c)
11:        then
12:            c_ID = c;
13:        end if
14:    end if
15:    send ("done-mig", n, cma);
16: end procedure
```

`NotifyDoneGeneration` (algo. 1 step 12), is listed as algo. 4 lines 19–23. Each cm-agent[7] sends "`done-gen`" to all other cm-agents. Then, each cm-agent sends "`attempt-mig`" to its member node-agents. Finally, either another generation begins or the evolutionary process terminates.

`NotifyDoneMigration` (algo. 1 step 9) is listed as algo. 4 lines 1–5. When a cm-agent receives a "`done-mig`" from all its member node-agents, it then sends[7] "`done-mig`" to all other cm-agents. `CoalitionSnapshot` (algo. 1 step 10) is listed as algo. 4 lines 6–15. The node-agents each send[7] their `node-ID` and `load` to their cm-agent. When a cm-agent receives these items from all its member node-agents, it constructs two new lists, the member-list (`mem-list`) and the member load-list (`ld-list`). These lists are sent to the other cm-agents. When each cm-agent receives all lists, they then send them to the member node-agents. `Elect` (algo. 1 step 11) is listed as algo. 4, lines 16–18. Each existing cm-agent[7] begins an election of a new cm-agent for the members of the new coalition (Bully algorithm (Garcia-Molina, 1982)).

Algorithm 4: Second-level: DistributedEvolveCoalitions Components.

```
1: procedure NOTIFYDONEMIGRATION
2:     ∀cma ∀n ∈ cma recv ("done-mig", n, cma);
3:     ∀cma_f ∀cma_t send ("done-mig", cma_f, cma_t);
4:     ∀cma_f ∀cma_t recv ("done-mig", cma);
5: end procedure
6: procedure COALITIONSNAPSHOT
7:     ∀cma ∀n ∈ cma send (node-ID, load, n, cma);
8:     ∀cma ∀n ∈ cma recv (node-ID, load, n);
9:     ∀cma
10:    ConstructLists (mem-list, ld-list);
11:    ∀cma_f ∀cma_t send (mem-list, ld-list, cma_f, cma_t);
12:    ∀cma_f ∀cma_t recv (mem-list, ld-list, cma_f, cma_t);
13:    ∀cma ∀n ∈ cma send (mem-lists, ld-lists, cma, n);
14:    ∀cma ∀n ∈ cma recv (mem-lists, ld-lists, cma, n);
15: end procedure
16: procedure Elect
17:    ∀cma Bully ();
18: end procedure
19: procedure NOTIFYDONEGENERATION
20:    ∀cma_f ∀cma_t send ("done gen g", cma_t, cma_f);
21:    ∀cma_f ∀cma_t recv ("done gen g", cma_t, cma_f);
22:    ∀cma ∀n ∈ cma, send ("attempt-mig", n);
23: end procedure
```

4 EXPERIMENTS

This study's extensive experiments are a preliminary comparison of the DMCF-spatial and DMCF-GGA models. The metrics measured were (1) average coaliton throughput, and (2) overall search cost. They use a estimate of the communication cost. This section contains the configurations of all experiments, and the results of these experiments.

4.1 Configurations

The comparison consisted 2 sets of five series of experiments. The first set measured DMCF-spatial and the second set measured a similarly configured DMCF-GGA. Fig. 5 lists all the attributes common to both models. The series of five experiments consists of an experiment for each coalition size. The experiments were performed on a simulated Grid consisting of 500 nodes. This simulated grid is modelled after the existing DAS-3 (Distributed ASCI Supercomputer 3) grid. DAS-3's worst case latency between two major nodes is 0.7 msec. Most of this latency is due to physical fiber distance traveled. [9,10]

[9] A maximum throughput was found in relatively few generations if the experiments had the total size of the all arriving jobs uniform on 1000M±500M. So, a precise comparision of the models was not possible.

[10] $(19 \times 526) + (1 \times 10000) = 20000\ Mflops$

Attribute	Value
nodes in the Grid	500
coalition sizes for each series	[23, 33, 45, 83, 100]
migration threshold for each series	[0.05 - 1.20] incremented by 0.05
job type	cpu-intensive
node capacity	1000 Mflops per sec
total size of arriving jobs for every 20 nodes in the Grid	19 nodes @ 526±500 Mflops and 1 node @ 10000±500 Mflops
probability of migration, p	0.25
condition of termination	no change for 100 generations (the Core is attained)
metrics	total throughput and migration count
number of trials	30

Figure 5: The attributes and values of the DMCF-spatial and DMCF-GGA experiments.

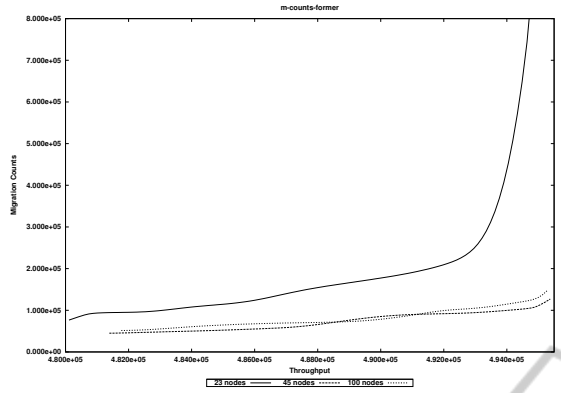
4.2 Results

This section contains the results of the following experiments: Phase I – Migration Threshold vs. Throughput and Migration Counts, Phase I Migration Counts vs. Throughput, Phase II – Load Sharing Counts, and Overall Performance.

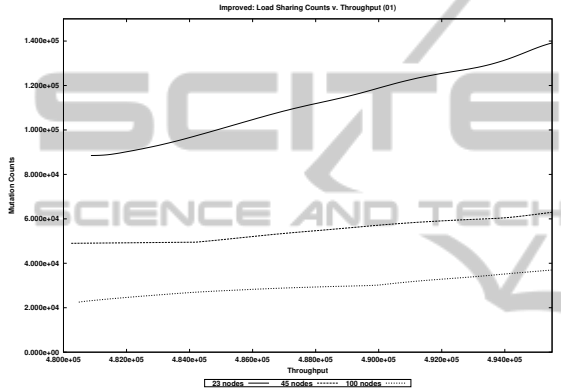
Phase I – Migration Threshold vs. Throughput and Migration Counts: The experiments showed that increasing the threshold, decreased the total throughput and decreased the migration counts. The migration threshold may be viewed as the *allowed error* in existing coalition throughput. For both models, a node may migrate if the average coalition throughput is above the threshold. Higher *allowed error* means fewer coalitions are candidates for migration. This causes both fewer migrations, and lowers the total throughput.

4.2.1 Phase I – Migration Counts vs. Throughput

The threshold selected to produce the following graphs has the fewest counts per throughput value. Fig. 6 depicts the migration counts compared to throughput for DMCF-spatial (fig. 6(a)) and DMCF-GGA model (fig. 6(b)). Both graphs have plots where coalition sizes are fixed at [23, 45, 100] (only 3 of 5 coalition sizes are shown). First, migration counts decrease as the coalition size increases. This may be because as the coalition size increases, the probability of finding underloaded nodes to compensate for overloaded nodes, increases. Hence, coalitions do not need to change and there are fewer migrations.



(a) DMCF-spatial model: the x-axis is the throughput, and the y-axis is the migration count.



(b) DMCF-GGA model: the x-axis is the throughput, and the y-axis is the migration count.

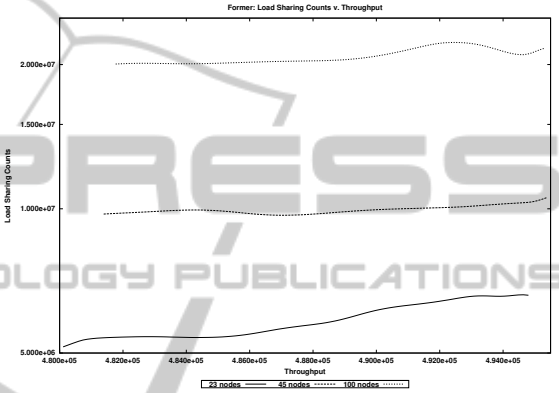
Figure 6: The two graphs compare the coalition formation search of the DMCF-spatial and the DMCF-GGA models. Each graph plots migration counts versus throughput for coalition sizes [23, 45, 100].

Secondly, both graphs show the migration counts increase as the throughput increases. This is likely due to an accumulation of migration counts as the generations proceed[11]. Also, DMCF-spatial has an exponential count increase for high throughputs if the coalition size is small (23). But DMCF-GGA shows the migration count increase over throughput is approximately linear for all coalition sizes. Finally, the graphs show the DMCF-GGA model has migration counts that are significantly smaller than DMCF-spatial (for all throughput values); >49% fewer if the coalition size of 23, >42% fewer for a coalition size of 45, and >43% for a coalition size of 100.

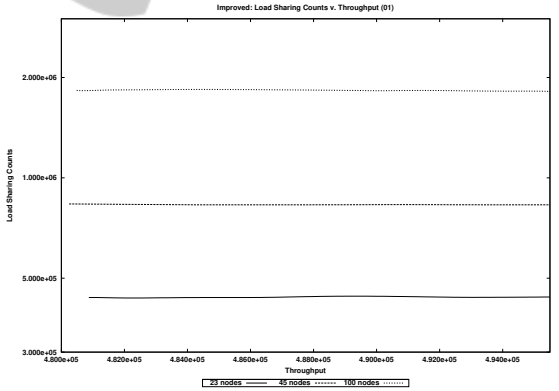
[11]Throughput is monotonically increasing from one generation to the next.

4.2.2 Phase II – Load Sharing Counts vs. Throughput

Fig. 7 shows load sharing counts compared to throughput for DMCF-spatial (fig. 7(a)) and DMCF-GGA (fig. 7(b)). For each coalition size (23, 45, 100), the DMCF-GGA model has considerably fewer state searches compared to DMCF-spatial (>17%). Also, the load sharing count among coalition members does not change as throughput increases. This is probably due to the implementation of FF as an exhaustive search. But, it will be improved in our future work.



(a) DMCF-spatial model: the x-axis is the throughput and the y-axis is the load sharing state change count.



(b) DMCF-GGA model: the x-axis is the throughput and the y-axis is the load sharing state change count.

Figure 7: The two graphs compare local load sharing within the coalition of the DMCF-spatial and the DMCF-GGA models. Each graph plots state change counts versus throughput for coalition sizes [23, 45, 100].

4.2.3 Overall Performance

4.2.3.1 Comparison of DMCF-GGA to Existing Methods. We compared the results of the DMCF-GGA experiments to three existing methods (1) first come first served (FCFS), (2) FCFS first fit (FCFS-FF), and (3) round robin (RR). The first method,

FCFS, is Condor's scheduling method. If FCFS is utilized on a grid of 500 servers, the total throughput is 2.74e5. DMCF-GGA improves total throughput by 80% over FCFS for a coalition size of 23. The second method, FCFS-FF, uses FCFS to form coalitions and then it uses FF to load share within each coalition. The results are shown in fig. 8(a). DMCF-GGA has a 19% improvement for a coalition size of 23. The results of the third method, RR, are depicted in fig. 8(b). Though DMCF-GGA's throughput gains are small, DMCF-GGA uses 50% fewer state changes. Moreover, DMCF-GGA forms coalitions that require less load sharing than RR.

4.2.3.2 DMCF-GGA Cost (Seconds). Figs 9(a) and 9(b) lists the cost of DMCF-GGA Phase I, fig. 9(c) has Phase II, and their total – the overall cost – is provided by fig. 9(d). The propagation delay used in figs. 9(a) – 9(c) is $counts$[12] $\times$ 0.7 msec[13]. Eqn. 1 calculates the Phase I cost, where α is the Phase I overhead cost given by fig. 9(a), 225 is the average number of generations to attain the Core, and β is the migration counts given by fig. 9(b).

$$cost_{Phase_I} = \alpha \times 225 + \beta \quad (1)$$

The total DMCF-GGA cost, $cost_{total}$, is calculated by eqn. 2, where $cost_{Phase_{II}}$ is the cost of FF (fig. 9(c)). Fig. 9(d) provides the totals.

$$cost_{total} = cost_{Phase_I} + cost_{Phase_{II}} \quad (2)$$

coalition size	FCFS with FF	DMCF-GGA	Percent Improvement
23	4.15e5	4.95e5	19
45	4.29e5	4.99e5	14
100	4.61e5	4.99e5	8

(a) Maximum throughput of FCFS-FF vs. DMCF-GGA

coalition size	Round Robin	DMCF-GGA	Percent Improvement
23	4.63e5	4.95e5	8
45	4.62e5	4.99e5	8
100	4.94e5	4.99e5	1

(b) Maximum throughput of Round Robin vs. DMCF-GGA

Figure 8: Comparison of DMCF-GGA and existing solutions in terms of total throughput.

4.2.3.3 Cost/Benefit Observations. The number of generations is seen to have a large effect on the overhead (fig. 9(a)). The overhead has a greater effect on Phase I cost than migration's states searched for the given coalition sizes ($6.3 \times 225 = 1417ms$ vs. 9(b)). This may not be the case if coalitions smaller than 23 are used. Phase I has a 64% greater effect on total cost than Phase II for coalition size of 23, (figs. 9(a), 9(b), and 9(c)). Hence, FF is not a bottleneck for small coalitions, but it is one for the larger sized coalitions. A goal in our future work is to find an exact upper bound for coalition sizes.

Fig. 9(d) (the result of 2) shows coalition size to have a greater effect on search cost than throughput. Eqn. 3 gives the sec/(coalition size) relation.

$$y = 0.023x\frac{sec}{size} + 2.213 \quad (3)$$

Though this suggests small coalitions reduce search cost, migration counts increase as the coalition size decreases. So, we also need to find an exact lower bound for coalition sizes. In addition, the results of fig. 9(d) imply for DMCF-GGA to run at 1% of the execution time the job duration is 221.3 (sec) for a coalition size of 23, 266.3 (sec) for a coalition size of 45, and 399.4 (sec) for a coalition size of 100.

Procedure	Counts	Overhead Cost (ms)
ConditionalMigration	1	0.7
NotifyDoneMigration	1	0.7
CoalitionSnapshot	3	2.1
Elect	2	1.4
NotifyDoneGeneration	2	1.4
Totals for Phase I	9	6.3

(a) Overhead cost (cost of communication) per generation for Phase I procedures.

	Counts	Migration Cost (ms)
throughput 980	239,115, or 62	167, 80, 43
throughput 985	253,120, or 67	177, 84, 47
throughput 990	275,126, or 74	192, 88, 52

(b) Cost (ms) for Phase I – Migration – coalition sizes of 23, 45, and 100

	Counts	FF Cost (ms)
all throughput levels	884, 1660, 3620	618, 1162, 2534

(c) Cost (ms) for Phase II – First Fit – coalition sizes of 23, 45, and 100

	Total cost for each coalition size (sec)		
	23	45	100
throughput 4.90e5	2.202	2.659	3.995
throughput 4.92e5	2.213	2.663	3.999
throughput 4.95e5	2.227	2.668	4.004

(d) Total search cost (sec) for different coalition sizes.

Figure 9: Search cost (sec) for DMCF-GGA Phases I and II, including the effect of the network.

[12] migration search counts

[13] 0.7 msec is DAS-3's worst case node to node latency. DAS-3 has direct optical links between nodes. A congested link does not have a large effect on latency.

5 CONCLUSIONS AND FUTURE DIRECTIONS

First, an important result of this study is the search cost of both migration and local load sharing of DMCF-GGA is less than DMCF-spatial by a factor of 10. Also, DMCF-GGA outperforms the controls, FCFS, FCFS-FF and RR (sec. 4.2.3.1). Thus, DMCF-GGA may be a candidate for use as a scheduler in Condor. Secondly, it has determined the linear relation between coalition size and search cost for high throughput. And, we have found preliminary estimates for the lower and upper bounds of the effective coalition size. Further, we have found the average job sizes required for DMCF-GGA to run at 1% of the job execution time.

In our future work, optimizing Phase I must be a priority since as the model scales up to 50,000, the Phase I cost scales increases 100:1. Given that the number of generations and counts for migration are the main factors for the delay, improving the search precision (e.g. adding a bulk migrate) could reduce the delay. A bulk migration could be defined as 20% of a coalition's nodes migrating at the same time. Also, currently migration may get stuck at a local maximum for some cases[14]. Bulk migration may prevent this.

Generally, it seems that the difficulty of the problem (e.g. job size composition) has a large effect on both the number of generations and the states searched during migration. Finding a precise correlation between problem difficulty, and these two factors is another goal.

For the remainder of our work plan we envision the following items: (1) To make the model more realistic, jobs should be non-divisible. (2) Since, for DMCF-spatial, coalitions may overlap, the data about coalition composition is unclear. But, study of DMCF-GGA coalition composition in detail may offer insight about conditional search. Specifically, finding how job size compositions and coalition compositions affect the relation between coalition size and search cost. (3) Restructure the model so it can encompass multicore nodes. (4) Performance test DMCF-GGA within the SimGrid framework. This framework enables the simulation of applications in a distributed computing environment for controlled development and evaluation of the algorithms. (5) Matchmaking (Raman et al., 1998) is a component of Condor, and we will enhance it with the DMCF-GGA algorithm.

[14]because it may terminate after no change for 100 generations, and a change may occur after 100 generations.

ACKNOWLEDGEMENTS

I would like to thank Jae C. Oh, Dmitri E. Volper and Judy Qiu for their suggestions and support.

REFERENCES

Brucker, P. (2004). *Scheduling*, chapter Computational Complexity, pages 50–60. Springer, Osnabruck, Germany, 4th edition.

Csari, B., Monostori, L., and Kadar, B. (2004). Learning and cooperation in a distributed market-based production control system. In *Proceedings of the 5th International Workshop on Emergent Synthesis*, pages 109–116.

Decker, K., Durfee, E., and Lesser, V. (1998). Evaluating Research in Cooperative Distributed Problem Solving. *UMass Computer Science Technical Report 88-89*.

Fiala, J. and Paulusma, D. (2005). A complete complexity classification of the role assignment problem. *Theor. Comput. Sci.*, 349:67–81.

Foster, I. and (Eds.), C. K. (1999). *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers.

Garcia-Molina, H. (1982). Elections in a distributed computing system. *IEEE Trans. Comput.*, 31:48–59.

Hovey, L., Volper, D. E., and Oh, J. C. (2003). Adaptive dynamic load-balancing through evolutionary formation of coalitions. In Abraham, A., Koppen, M., and Franke, K., editors, *Design and Application of Hybrid Intellient Systems*, pages 194–203, Ohmsha. IOS Press.

Ibaraki, T. and Katoh, N. (1988). *Resource Allocation Problems: Algorithmic Approaches*, chapter 1, pages 1–9. MIT Press, Cambridge, MA, USA.

Kowalski, R. and Sadri, F. (1996). Towards a unified agent architecture that combines rationality with reactivity. In Pedreschi, D. and Zaniolo, C., editors, *Logic in Databases*, volume 1154 of *Lecture Notes in Computer Science*, pages 135–149. Springer Berlin / Heidelberg. 10.1007/BFb0031739.

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97.

Livny, M., Basney, J., Raman, R., and Tannenbaum, T. (1997). Mechanisms for high throughput computing. *SPEEDUP Journal*, 11(1).

Michalewicz, Z. (1999). *Genetic Algorithms + Data Structure = Evolution Programs*, chapter 11, pages 251–253. Springer-Verlag, New York, New York.

Oliphant, M. (1994). Evolving cooperation in the non-iterated prisoner's dilemma: The importance of spatial organization. In Brooks, R. and Maes, P., editors, *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 349–352. MIT Press.

Pisinger, D. (1999). An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114:528–541.

Raman, R., Livny, M., and Solomon, M. (1998). Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, pages 28–31, Chicago, IL.

Sandholm, T. (1999). Distributed rational decision making. In Weiss, G., editor, *Multiagent Systems. A modern approach to distributed artificial intelligence*, volume 1 of *Reviews in important subjects*, chapter 5, pages 241–251. The MIT Press, Munich, Germany.

Vazirani, V. V. (2004). *Approximation Algorithms*, chapter 1, pages 1–2. Springer.

Weichhart, G., Affenzeller, M., Reitbauer, A., and Wagner, S. (2004). Modelling of an agent-based schedule optimisation system. In *Proceedings of the IMS International Forum*.

Wu, A. S., Yu, H., Jin, S., and Lin, K.-C. (2004). An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Trans. Parallel Distrib. Syst.*, 15(9):824–834. Member-Schiavone, Guy.