AGENT BASED MATCHMAKING

MASTER OF SCIENCE THESIS
IMIT/KTH 2003-2004

BY
NITIN PANCHANATHAN

Acknowledgements

The completion of this thesis has been highly dependent on the good will and support of many people. Words cannot express how much I want to thank the following people who have put in valuable time and effort in guiding me despite their hectic schedule and other responsibilities.

Thomas Sjöland: My Examiner at KTH(Sweden).
Luc Onana Alima: My Supervisor at KTH(Sweden).
Koen Bertels: CEO Upsilon Research and Professor at the department of Computer Engineering at Delft and my supervisor too.
Elth Ogston: Doctoral Student at Vrije Universitat Netherlands.

Abstract

This thesis presents some suitable mechanisms to solve the centralized matchmaking problem. Matchmaking is done in order to fulfill the needs of agents, trading in the market place or in a grid-based environment. Agents have a particular task to accomplish for which they need resources. In our case, producer agents provide the resources and consumer agents want to perform tasks. With the help of suitable mechanisms the market matches resources and tasks. We have developed a simulation environment to evaluate the mechanisms we have built. We have also studied how efficient these mechanisms are in terms of matching time and resources allocated. We found first match to be the simplest and fastest in terms of matching time. Minimum difference was the most efficient of them all, in terms of resource allocation. Minimum distance was the slowest in terms of matching time. We performed further experiments to move towards the peer-to-peer approach using partitioning. We found that the efficiency plots for first match and minimum difference indicate that centralized approach always results in better efficiency compared to partitioning approach. The minimum distance approach is the most suitable one when partitioning was used, as the efficiency is almost the same when compared to centralized approach.

Table of Contents

1. Introduction

"True knowledge exists in knowing that you know nothing and in knowing that you know nothing, that makes you the smartest of all".

- Socrates

How do we match resources and tasks of agents trading in the market? The answer to this is to write some complex programs that would solve the above problem. The procedural approach is used to solve such a problem. In procedural approach the problem is first decomposed in order to simplify it so that it can be solved. Such decomposition is not always possible because matchmaking problems are too intricate, widely distributed and have a lot of ambiguity in them. We use a simple agent based approach in order to tackle this problem. In this approach each of these agents are assigned small problems so that they can on their own coordinate and then a solution for the complex problem can be found. Two things have been considered while using agent-based approach, first, the individual complexity of the agents and second, the amount of centralized control in the system. The agents that participate in the matchmaking are simple and use up minimum resources. These kinds of agents are also called minimal agents. Minimal agents perform matchmaking either at the centralized directory or market place. Much care needs to be taken for the design of centralized directory and minimal agents to see that it is kept as simple as possible.

1.1 Structure of report

The report is structured as follows.

In this chapter we discuss the motivations behind using the agent based approach over procedural approach. In chapter 2 we specify clearly what the matchmaking problem is and also talk about work done by others and how it is similar or dissimilar to what we have done. In Chapter 3 we discuss the model, which would suitably solve this problem. Having built the model, we then proceed further and build a simulation based on the model and perform some experiments. Chapter 4 discusses the experiments done and also the extensions that we have made to the basic experiment. Many more additions could be done to the model to make it more robust and realistic. We provide suggestions and also conclude by providing the results, which we obtained. Chapter 5 discusses these issues. Chapter 6 provides all the references that were used.

1.2 Motivation for using agent based approach over procedure based approach

The usual approach towards handling complex tasks is the procedure-based approach. But agent based approach has its advantages over the procedure based approach. One of the advantages is that code can be maintained easily [1]. When we think in terms of parallel and distributed computing this plays an important role in making these possible [1]. We also have robustness and increased performance on using this approach [1]. When interactivity is needed it is best to use this approach [1]. Each of the agents can be assigned a role and better co-operation is seen on using this approach. They also provide services of different kinds to a particular community [2]. These are some of the factors that motivate one to follow an agent-based approach.

2 Problem statement and related research

2 .1 Problem statement

We are looking into resource allocation and task distribution in a grid based environment. Grids are a set of interconnected nodes. Agents run on these nodes. Nodes could have either resources or tasks available with them. Allocation of resources and tasks need to be done using agents. In such a scenario we encounter sub-optimal allocation of resources and tasks. This happens because some nodes have too many tasks and some nodes may have extra resources. What is needed is task re-distribution and proper resource allocation. We don't need certain nodes with extra resources not having any tasks that would use up the resources. Similarly there should not be too many tasks are at one node overburdening it. We investigate matchmaking functions that are suitable for solving such problems in the centralized scenario .We then move towards the partitioning approach and finally compare both these approaches in terms of their efficiency.

2.2 Related research

Resource allocation is needed when there are many tasks, which are not accomplished due to the lack of resources. Similarly task allocation is needed when there is abundance of idling resources with no tasks to use them up. Allocation is also useful in grid like environments where access to grid based components is needed[3]. Matchmaking is done to rebalance the workload. The question now arises as to how matchmaking has to be done. There are basically three methods to aid in matchmaking. First scheme uses middle agents providing some kind of central directory[8][4][3]. An alternative approach is to use market bidding mechanisms where bids and offers are broadcasted to all agents[14][6][7]. A third possibility is to allow peer–peer communication that includes communication cost but using only local information [10][13]. We extend the work on matchmaking of minimal agents with and without a facilitator as described in [10][11][12].

It investigated matchmaking with and without a centralized facilitator by using an auction based mechanism matchmaking achieves 93% success rate for population upto 32k agents. To achieve such a success rate it might need upto 300 bidding rounds.
The advantage of this is that the communication overhead is considerably reduced. This happens because no broadcasting is required. However the time required to find a match increases considerably.

The main contribution in this thesis is as follows.

- The simplest matchmaking function is the most efficient one both in terms of resource usage, task execution and matching time.
- More computationally expensive functions only yield a marginal improvement in the order of 1% as far as the resource usage and task execution are concerned.
- Centralized matchmaking is scalable in the range of (studied) 5 to 50K agents interacting on the market.
- Multiple matchmakers can be introduced to reduce substantially the communication overhead. It is empirically shown that a population size should never be lower than 10K or larger than 20K.

3 Our approach

Having seen some of the approaches used by others in the area of agent based matchmaking we now explain our model and also justify why we chose the centralized approach over other approaches. We start with the description of the model.

3.1 Model

The goal of this experiment is to evaluate different ways of redistributing tasks to different processing nodes, given some constraints. We consider a grid like environment where some nodes might fall idle whereas others are still overloaded. The latter category wants to delegate some of the tasks in their local queue to other nodes in order to decrease the overall computing time. We assume 2 categories of agents, producers and consumers. Producers have processing power to provide resources and consumers are looking for additional processing power to execute the tasks in their queue. The goal is to facilitate the match making process in such a way that the maximum number of tasks is executed, given the available resources. This implies that a consumer has to find a producer that can provide sufficient processing time to execute the task.

The matchmaking mechanism involves a centralized mechanism (similar to, but different from a market) that will receive from producers the resources offered and from the consumers the requested tasks. No subsequent bidding or negotiation is then required.

We make the following assumptions:

- We assume that each agent is directly and with the same average latency connected to the matchmaker and that the cost for this connection is the same for each agent. This simplifies the problem and even allows us to exclude it from the analysis.

- Tasks are atomic by nature and cannot be divided. Each task has a particular complexity which is represented by an integer value. This value indicates the amount of resources required in order to be executed. Resources are represented in a similar way. Whether or not a particular task can be executed by an available resource is then simply determined by comparing these integer values.

- One consumer can match with only one producer who has sufficient resources to execute the task. This means that we currently do not look at collaborative issues.

- We generate randomly the initial task and resource allocation from a uniform distribution.

- We are currently using a simulation model in which the agent requests are treated in a pure sequential way. In a realistic setting, the match making would occur in an asynchronous way where the different offers and bids are treated as they are submitted.

- We assume that there are an equal number of consumers and producers. This is not restrictive as the random generation of tasks and resources can result in a zero value which is similar to taking the agent out of the population.

- When referring to 'population', we mean the number of agents that use the matchmaker to buy or sell processing time. They represent only a fraction of the actual population.

In our environment, we have N agents: $A = \{a1 \ldots\ldots an\}$.Some of these agents, called consumers, has tasks to perform $Ta = \{t1 \ldots\ldots tk\}$ for which they are looking for additional resources and others, called producers, have resources to sell, $Ra = \{r1 \ldots\ldots rk\}$. There is a matchmaker to which both consumers and producers announce their requests. Consumers will send to the matchmaker the number of tasks they want to delegate and the producer will announce in the same way how much processing time it has available.

The matchmaker then uses that information to match consumer requests to producer bids.

The basic matching goes as follows: $f : C * P \to [0:1]$ with
$f(ci: pj) = 1:$ if $(ci; pj)$ is a matching pair
$0:$ otherwise.

We define 4 matching functions that vary in complexity and information used for matching. Evidently, one can define any number of matching functions taking for instance taking into account the Quality of Service into account or any other relevant factor. The choice of our functions is justified in the sense that we want to compare an extremely simple one (FirstMatch) with functions that are more complicated as they take more information into account to compute a match. The chosen functions are defined as follows.

Sorting:

Sort the producer requests and consumer requests as and when they arrive in descending order and matches the consumers and producers respectively by processing the queue sequentially.

FirstMatch :

For each consumer request, the matchmaker matches the consumer to the first producer that has enough resources to execute the consumer's request.

MinDifference :

The consumer is matched with that producer that has enough resources but also that yields the lowest difference between the requested task and available resources. This approach attempts to minimize the unused resources. We emphasize that we do not sort the producers or consumers data.

MinDistance :

The consumer is matched not only when enough resources are available but also tries to minimize the distance between the two paired nodes. This is to minimize as much as possible the distance that has to be traveled between the two paired nodes. As each node has an (x,y)-coordinate, we compute the Euclidean distance between two nodes. Similar to Min-Difference, the producers or consumers are not sorted using their (x,y)-coordinates.

3.3 Justifications for centralized matchmaking

Having seen the model one may wonder as to why we chose to use the centralized approach and not other approaches that were discussed in the previous section. We provide a few convincing reasons as to why centralized match making was used.

Agent properties:

In [11], agents are actively bidding in an auction. This implies that they react to bids and offers made by other agents. On the basis of past experiences, the agents learn to optimize their bidding strategies and require a learning algorithm. In the context of minimal agents, we do not want to include such bidding or learning mechanisms as it creates too much overhead. In [10], peer to peer local interaction was induced by allowing agents to interact locally with neighboring agents. These interaction patterns can be changed whenever no match is occurring. This induces quite some overhead and multiple searches before a good match occurs. Our agent is simpler as its only communication means (as far as matchmaking is concerned) is directly with the matchmaker.

Bidding rounds and messages transmitted:

An important aspect is the number of messages that need to be transmitted over the network in order to reach a particular allocation state. Our approach requires that we need only a minimum of 2 messages per agent where tasks are requested and resources are allocated. Similar to [11], this implies that the time complexity of our approach is O(N). By introducing multiple matchmakers or auctions, this message distribution is reduced to O(log(N)). In [10], a related albeit different problem of agent collaboration is investigated. Agents have to produce an optimal solution given some global objective. The general distributed constraint optimization assumes that each agent is sending information to all of its linked descendants. Such an approach is known to have exponential time($O(2^N)$) and is only feasible for a very low number of agents. In conclusion, we wanted to avoid bidding or any other form of message passing other than submitting a request (bid or offer) to the matchmaker. This avoids additional message broadcasting that need to be sent to all other agents informing them of the bid made by any of the other agents.

Matchmaking:

A third aspect of our approach involves the actual match making process. In [10][12], localized random search is used to find a match. In [11], sorting is required for matchmaking as the highest bid is matched to the lowest offer, etc. Given the large population sizes we are interested in, we avoid the use of such expensive mechanisms. Sorting for instance involves queuing and searching also requires resources or tasks to be stored before hand.

Time complexity:

Since we avoid as much as possible expensive operations, the theoretical estimation of the best case complexity is O(log N), the average case is O(N) and the worst case is *O*(*N*^2). From our experiments which will be discussed later in the paper, we arrive at the conclusion that for smaller population i.e up to 20000 we have O(log N) behavior, which implies that the approach is potentially scalable. Beyond 20000 population we have complexities varying between O(log N) to *O*(*N*^2).

3.4 Why build matchmaking functions based on some optimization criteria?

When matchmaking needs to be done between a set of producers and consumers no suitable general solution exists to find all possible solutions for the given sets. This is illustrated with the help of suitable example.

We are given sets

C: non empty non zero and non negative set of consumers. Each consumer has a need. This could be a need from any arbitrary consumer.

P: non empty non zero and non negative set of producers. Each producer has a need. . this could be a need from any arbitrary producer

Constraints:

The basic criteria are that no two consumers should be paired with the same producer and no two producers should be paired with the same consumer. The other criteria are that the producer resource should be always greater than and equal to the consumer resource. Otherwise the pairing between the producer and consumer cannot be done

Objective:

We need to find a set of pairs of all producers and consumers such that producers satisfy the constraints put forward by the consumers.

Producers

P1	P2	P3	P4	P5
5	2	3	2	1

Consumers

C1	C2	C3	C4	C5
1	2	2	1	1

Solution 1

(P1,C1),(P2,C2),(P3,C3),(P4,C4),(P5,C5)

Solution 2

(P1,C2),(P2,C1),(P3,C3),(P4,C4),(P5,C5)

Solution 3

(P1,C3),(P2,C1),(P3,C2),(P4,C4),(P5,C5)

Solution 4
……………………………………………….

Solution 5
……………………………………………….

There are many such solutions possible.

I made an attempt to write an algorithm to accomplish the task as specified above. I was able to find a solution by generating all possible combinations of consumer for each producer. But this was not scalable and was of the exponential type (2^n) by nature. This had become a combinatorial problem, which I had to solve using some suitable optimization criteria. We decided to choose some simple optimization criteria, which motivated us to create first match, minimum difference and minimum distance. All these algorithms scale well and find suitable solutions if not all possible solutions.

3.5 Matchmaking mechanisms

To aid in the matchmaking processes between suitable producers and consumers there have been many criteria to decide which of the consumers should be paired with which of the producers. The basic criteria are that no two consumers should be paired with the same producer and no two producers should be paired with the same consumer. The other criteria are that the producer resource should be always greater than and equal to the consumer resource. Otherwise the pairing between the producer and consumer cannot be done. There are three schemes, which have been used to do this. They are namely firstmatch, minimum difference function and minimum distance function. Each of these functions are explained in detail.

Though we have discussed sorting in brief we don’t choose to use that approach because it involves an additional overhead before matchmaking is actually done which we feel is unnecessary.

First Match function:

We are given sets

C: non empty non zero and non negative set of consumers. Each consumer has a need. This could be a need from any arbitrary consumer.

P: non empty non zero and non negative set of producers. Each producer has a need. . this could be a need from any arbitrary producer

Objective:

We need to find a set of pairs of producers and consumers such that producers satisfy the constraints put forward by the consumers.

Principle:

We take each consumer and check with each producer in the producer list if match occurs or not. As soon as a match occurs we remove the matched producer and consumer pair and continue to iterate over the new producer list and consumer list to find the next match. If there is no match we choose the next consumer and check with the producer list. This process is repeated till either of these lists are empty.

Example:

Suppose we have a set of producers and consumers say 5 producers and Consumers. Let us represent Producers as P1,P2,P3,P4 and P5. Let us represent Consumers as C1,C2,C3,C4, and C5. We will randomly assign values to both of them as follows.

Producers

P1	P2	P3	P4	P5
5	2	3	2	1

Consumers

C1	C2	C3	C4	C5
1	2	2	1	1

The possible matches for each consumer is listed below.

For C1: The possible matches are P1, P2, P3, P4 and P5.
As soon as we find a match which is P1. We assign it to C1. We remove the corresponding producer and consumer from the list.

For C2: The possible matches are P2, P3, P4 and P5.
As soon as we find a match which is P2. We assign it to C2. We remove the corresponding producer and consumer from the list.

For C3: The possible matches are P3, P4 and P5.
As soon as we find a match which is P3. We assign it to C3. We remove the corresponding producer and consumer from the list.

For C4: The possible matches are P4 and P5.
As soon as we find a match which is P4. We assign it to C4. We remove the corresponding producer and consumer from the list.

For C5: The possible matches are P5.
As soon as we find a match which is P5. We assign it to C5. We remove the corresponding producer and consumer from the list.

The Allocation, which was achieved using this scheme, was

(P1,C1), (P2,C2), (P3,C3), (P4,C4), (P5,C5).

ALGORITHM FOR FIRST MATCH:

DEFINITIONS OF THE VARIABLES USED IN THE ALGORITHM

Match:

A mapping that takes the consumer c, then returns the producer that is allocated to c. This mapping is actually the output of the algorithm. Initially this mapping is an empty set of pairs.

R:

A mapping that takes the producer p, then returns the resources at producer p.

T:

A mapping that takes the consumer c, then returns the tasks of consumer c.

C:

A variable used to the capture the CURRENT set of customers, Initially this variable is set to all consumers.

P:

A variable used to the capture the CURRENT set of producers, Initially this variable is set to all producers.

The function SelectOneFrom:

This takes current set of customers then returns one consumer selected evenly At random.

FirstCandidateFor:

A mapping that takes a consumer c, then returns first producer that satisfy the needs of consumer c.

Given a Set A,I use the notation A\(x) to mean that the element x is removed from the set A.

FIRST MATCH ALGORITHM

```
While C!=emptyset do
                    c:=SelectOneFrom(C)
While P!=emptyset do
                    p:=SelectOneFrom(P)
Match(c):=c in FirstCandidatesFor(P) such that {R(p) >=T(c)}

C:=C\{c}

P=P\{Match(c)}


end while {P}
end while{C}
```

TIME COMPLEXITY:

Worst case:

O(N^2)

Best case:

O(log N)

Average case:

O(N)

Difference function:

Given an input of non negative non zero producers and consumers the function evaluates for every consumer the possible producer matches. It then finds the difference between the single consumer and possible producer matches. The producer with the lowest difference is selected and paired with the corresponding consumer.

Principle:

We take each consumer and check with each producer in the producer list, the difference in their values and then find the minimum difference between the consumer and producer. We find the producer-consumer pair that satisfies the minimum difference between their value criteria and also the condition that the producer value should be greater than equal to the consumer value. As soon this criterion is satisfied we remove the matched producer-consumer pair from their respective lists and continue with the same procedure. In case there was no match we take the next consumer and start matching with the available producer list. This process is repeated till either of these lists are empty.

Example:

Suppose we have a set of producers and consumers say 5 producers and Consumers. Let us represent Producers as P1,P2,P3,P4 and P5. Let us represent Consumers as C1,C2,C3,C4, and C5. We will randomly assign values to both of them as follows.

Producers

P1	P2	P3	P4	P5
5	2	3	2	1

Consumers

C1	C2	C3	C4	C5
1	2	2	1	1

The possible matches for each consumer are listed below.

For C1: The possible matches are P1, P2, P3, P4 and P5.
We then calculate the difference between C1 and all the corresponding matched producers. They are (P1-C1), (P2-C1), (P3-C1), (P4-C1), (P5-C1). Out of these differences the one with the minimum difference is selected so we select (P5,C1). We removed the selected producer, consumer pair.

For C2: The possible matches are P1, P2, P3 and P4
We then calculate the difference between C2 and all the corresponding matched producers. They are (P1-C2), (P2-C2), (P3-C2), (P4-C2), Out of these differences the one with the minimum difference is selected so we select (P2,C2). Though (P4,C2) pair also results in the minimum difference we select the pair that results in the first match.
We removed the selected producer, consumer pair.

For C3: The possible matches are P1, P3 and P4.
We then calculate the difference between C3 and all the corresponding matched producers. That is (P1-C3), (P3-C3), (P4-C3). Out of these differences the one with the minimum difference is selected so we select (P4,C3). We removed the selected producer, consumer pair.

For C4: The possible matches are P1 and P3.
We then calculate the difference between C4 and all the corresponding matched producers. That is (P1-C4), (P3-C4). Out of these distances the one with the minimum difference is selected so we select (P3,C4). We removed the selected producer, consumer pair.

For C5: The possible matches are P1
We then calculate the difference between C5 and all the corresponding matched producers. That is (P1-C5). We removed the selected producer, consumer pair.

The Allocation, which was achieved using this scheme, was

(P1,C5), (P2,C2), (P3,C4), (P4,C3), (P5,C1).

ALGORITHM FOR MINIMUM DIFFERENCE:

DEFINITIONS OF THE VARIABLES USED IN THE ALGORITHM:

Match:

A mapping that takes a consumer c, then returns the producer that is allocated
To C. This mapping is actually the output of the algorithm, initially this mapping
Is an empty set of pairs.

R:

A mapping that takes a producer p, then returns the resources at producer p.

T:

A mapping that takes a consumer c, then returns the tasks at consumer c.

C:

A variable used to capture the CURRENT set of consumers. Initially this is set of all consumers.

P:

A variable used to capture the CURRENT set of producers. Initially this is set of all producers.

The Function CandidatesFor:

A mapping that takes a consumer c, then returns the set of producers that satisfy the needs of the consumer.

The Function SelectOneFrom:

A mapping that takes the current set of consumers and returns one consumer selected evenly and randomly chosen.

Given a set A,I use the notation A\{x} to mean that the element x is removed from the set A.

RULES FOR NEED SATISFACTION:

Find all Match(c):=p in p such that {R(p)>=T(c)}
Find best match from all matches such that {R(p)-T(c)} is minimum

MINIMUM DIFFERENCE ALGORITHM

```
While C!=emptyset do
                    c:=SelectOneFrom(C)

While P!=emptyset do
                    p:=SelectOneFrom(P)



CandidatesFor(C):={p in P: R(p) >=T(c)}



Match(c):=p in CandidatesFor(c) such that
R(p)=Minimum{r(q):q in CandidatesFor(C)}

C=C\{c}

P=P\{Match(c)}

End while {P}
End while{C}
```

TIME COMPLEXITY:

Worst case:

O(N^2)

Best case:

O(N^2)

Average case:

O(N^2)

Weights function:

Given an input of non negative non zero producers and consumers the function evaluates for every consumer the possible producer distance given the cartesian coordinate values for X and Y assigned to all the consumers and producers. For every consumer the possible producer match is selected. The Euclidian distance between the respective coordinates is found out. From all possible suitable matches the pair is selected which has the lowest Euclidean distance.

Euclidean distance formula:
If we have two points (X1,Y1) and (X2,Y2). The distance is calculated by using the formula squareroot((X2-X1)^2 + (Y2-Y1)^2)).

Principle:

We take each consumer and check with each producer in the producer list the difference in their values and then find the minimum distance between the consumer and producer coordinate values. We find the producer-consumer pair that satisfies the minimum distance between their coordinate value criteria and also the condition that the producer value should be greater than equal to the consumer value.

As soon this criterion is satisfied we remove the matched producer-consumer pair from their respective lists and continue.

In case there was no match we take the next consumer and start matching with the available producer list. This process is repeated till either of these lists are empty.

Example:

Suppose we have a set of producers and consumers say 5 producers and Consumers.
Let us represent Producers as P1, P2, P3, P4 and P5. Let us represent Consumers as C1, C2, C3, C4, and C5. We will randomly assign values to both of them as follows. We also assign random values to x and y cartesian coordinates to the producers and consumers respectively.

Producers

P1	P2	P3	P4	P5
5	2	3	2	1

X-Coordinates

X1	X2	X3	X4	X5
2	3	2	1	4

Y-Coordinates

Y1	Y2	Y3	Y4	Y5
1	3	3	4	1

Consumers

C1	C2	C3	C4	C5
1	2	2	1	1

X-Coordinates

X1	X2	X3	X4	X5
3	4	2	1	1

Y-Coordinates

Y1	Y2	Y3	Y4	Y5
2	1	1	1	1

The possible match for each consumer is listed below.

For C1: The possible matches are P1, P2, P3, P4 and P5.

We calculate the Euclidean distance between all the matches and C1. They are respectively (1. 4), (1. 0), (1. 41), (2. 82), (1. 41). From this list (1. 0) is the minimum distance and this corresponds to (P2,C1) distance. So this chosen pair is removed.

For C2: The possible matches are P1, P3, P4 and P5.
We calculate the Euclidean distance between all the matches and C2. They are respectively (2. 0), (2. 8) and (4. 24). From the list (2. 0) is the minimum distance and this corresponds to (P1,C2) distance. So this chosen pair is removed.

For C3: The possible matches are P3,P4 and P5.
We calculate the Euclidean distance between all the matches and C3. They are respectively (2. 0), (3. 16). From the list (2. 0) is the minimum distance and this corresponds to (P3,C3) distance. So this chosen pair is removed.

For C4: The possible matches are P4 and P5.
We calculate the Euclidean distance between all the matches and C4. They are respectively (3. 0), (3. 0). From the list (3. 0) is the minimum distance and this corresponds to (P4,C4) distance. The first match is taken into consideration. So the other matches with similar distance is not taken into consideration when making use of minimum distance criteria. So this chosen pair is removed

For C5: The possible matches are P5. From the list (3. 0) is the minimum distance and this corresponds to (P5,C5) distance. So this chosen pair is removed.

The Allocation, which was achieved using this scheme, was

(P1,C2), (P2,C1), (P3,C3), (P4,C4) and (P5,C5).

ALGORITHM FOR MINIMUM DISTANCE:

DEFINITIONS OF THE VARIABLES USED IN THE ALGORITHM:

Match:

A mapping that takes a consumer c, then returns the producer that is allocated
To C. This mapping is actually the output of the algorithm, initially this mapping
Is an empty set of pairs.

R:

A mapping that takes a producer p, then returns the resources at producer p.

T:

A mapping that takes a consumer c, then returns the tasks at consumer c.

C:

A variable used to capture the CURRENT set of consumers. Initially this is set of all consumers.

P:

A variable used to capture the CURRENT set of producers. Initially this is set of all producers.

Coor(x):

A function that takes a producer p or a consumer c,then returns the coordinate values for it.

EuclideanDistance:

This takes a pair of co-ordinates for eg P(x1,y1) and C(x2,y2) and computes
The distance using the formula squareroot((x2-x1)^2+(y2-y1)^2))

The Function CandidatesFor:

A mapping that takes a consumer c, then returns the set of producers that satisfy the needs of the consumer.

The Function SelectOneFrom:

A mapping that takes the current set of consumers and returns one consumer selected evenly and randomly chosen.

Given a set A,I use the notation A\{x} to mean that the element x is removed from the set A.

RULES FOR NEED SATISFACTION:

Find all Match(c):=p in p such that {R(p)>=T(c)}
Find best match from all matches such that Euclidean distance is minimum.

MINIMUM DISTANCE ALGORITHM

```
While C!=emptyset do
                c:=SelectOneFrom(C)


While P!=emptyset do
                p:=SelectOneFrom(P)

CandidatesFor(C):={p in P: R(p) >=T(c)}

Match(c):=p in CandidatesFor(c) such that R(p)=Eucledian distance{Coor(p),Coor{r(q):q in
CandidatesFor(C)}}
```

```
C=C\{c}

P=P\{Match(c)}

End while {P}
End while{C}
```

TIME COMPLEXITY:

Worst case:

O(N^2)

Best case:

O(N^2)

Average case:

O(N^2)

Psuedocode explained:

The pseudo-code for all the match making mechanisms are listed in the appendix section. The readers can refer to it there. The explanation is given here to ensure continuity for the user.

We read the values into producer array and consumer vector. We initialize the state. This producer makes use of a small state machine. There are two states.

The states are -1 and 2 respectively. When the state is -1 always make sure that we go on parsing the consumer array and storing the parsed element from the consumer array at the zero' th location of the vector. Having selected the onsumer we call the method find_best_match. This method takes 3 parameters the count, which is nothing but the number of agents in each run, consumers and set of producers. This returns the index where the match has taken place. This method in turn makes use of the does_match.

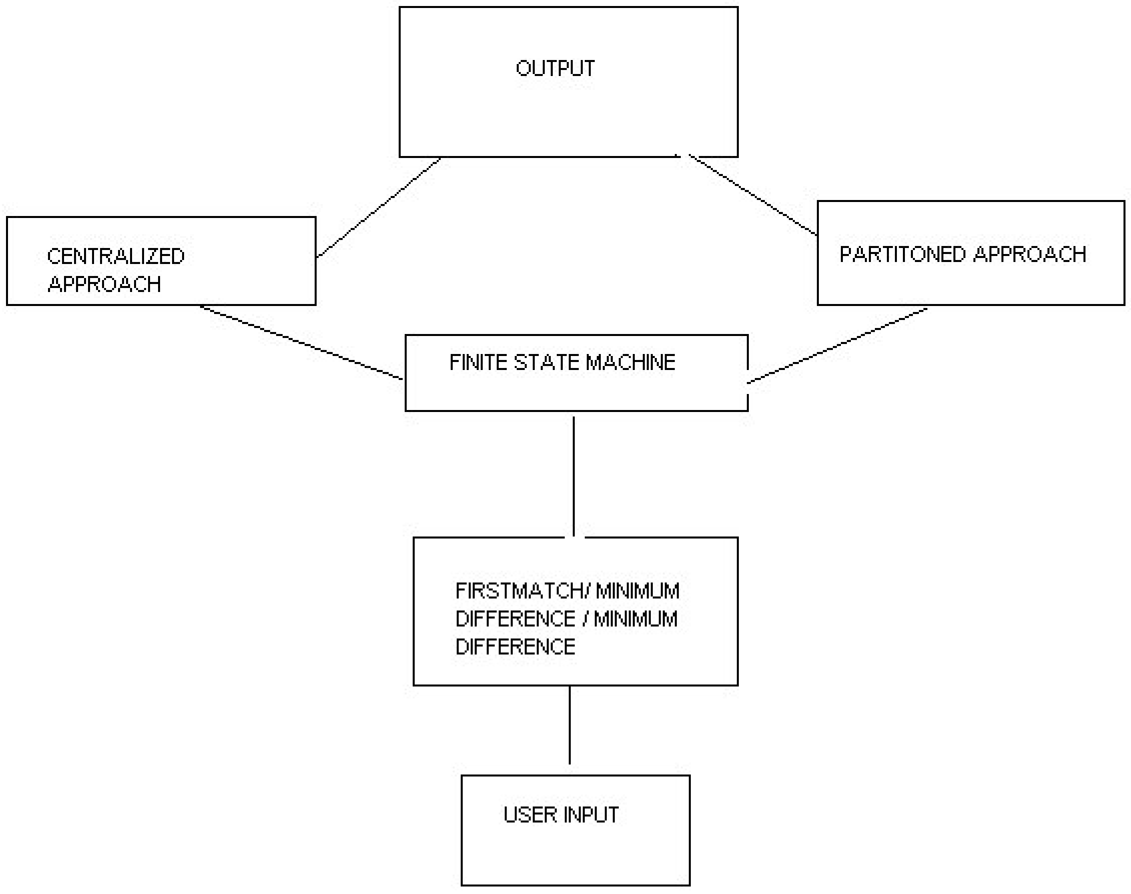
The does_match method actually has the criteria, which determines when a true match has occurred. This does_match method is called till all the producer elements have been checked with the consumer elements. This means checking needs to be done till end of producer vector.

The method sizeof(p) returns the size or the number of elements it currently holds. The while loop makes use of this within which the call to does_match is made. When the state is -2 which means we have found a match and we print it.
We keep toggling between both these states till all the consumers are compared with the set of producer arrays. When the toggling should be done is determined by the index parameter. When the index has reached the size of the producer array this means that we need to take the next consumer and check with all the producers. That is when state value is changed to -1.

3.6 Simulation

Architecture:

The simulation architecture is as shown in the diagram1. As soon as user decides on the number of agents that need to be used on the simulation, he can start the simulation. The control is transferred to the Finite state machine which is common for both the centralized approach and partition based approach (Will be discussed in the next chapter).The user can choose to run each of them separately or both of them together in order to compare the results.



SIMULATION ARCHITECTURE
(diagram 1)

Platform:

The simulation is built to be run on java platform. Any standard java compiler can be used to compile the source. We used jbuilder as an IDE to build the code and javac compiler was used for compilation:

Influence of the platform over test results:

The entire simulation architecture was built using java and the tests where performed on a standalone system. We chose an Intel Pentium IV System for all the tests. The results would not be any different on using any system that supported java. Since we measured parameters like matching time which is architecture dependent it would be interesting to see how the results would be on building a similar architecture using C language or in assembly language. This is left as future work. Since memory access and optimization can be performed to a greater extent using assembly and C language some of the parameters we measured here could vary. To what extent it would vary would be interesting to observe.

Parameters that user can change:

Centralized approach:

The number of agents that will be involved in the matchmaking [1..50000] for first match and [1..20000] for minimum difference and minimum difference.

Partition based approach:

Number of clusters e.g. 2,4,6.
Partitioning II to X-axis or Y-axis.

Output:

The result will be stored in .txt files. The users can then use it to plot results.

4 Experiments performed and empirical results:

4.1 First Match experiment:

What we wanted to investigate:

- Task execution efficiency:
  Percentage of allocated tasks of the different workloads.
- Resource usage efficiency:
  Percentage of available resources that are used up by the consumers.
- Matching time.

Set up of experiment:

We start performing the simulation starting at 5 agents and move towards the 50000 range. The number of agents increases by a factor of 10 with each run. Values assigned for resources and tasks. These values are generated randomly between an interval of [0. .10],[0. . 100] respectively.

Results:

A population of 5-50000 agents is used in the simulation. We then find out how many matches actually occurred. Then we plot a graph with the number of agents on the X-axis and time in milliseconds on Y-Axis. From figure 3 we can conclude that the matching time is proportional with the number of agents. There is a slow increase seen up to 20k. For this range the time taken lies between 0.2 milliseconds to 0.9 milliseconds. On going beyond the 20K boundary there is a rapid increase in the matching time. But this increase is less proportion. The allocation rate is less for very small population like 6K. The allocation percentage increases beyond 20k where it reaches the peak of 99. But for less than 20k it remains at 97-98. We have also used first difference, which would help us in calculating the variance. The variance is about 0.244 for a population size that is less than 20k. For sizes above 20k the variance is 11. 992. This can be observed from figure 4.

Graphs:

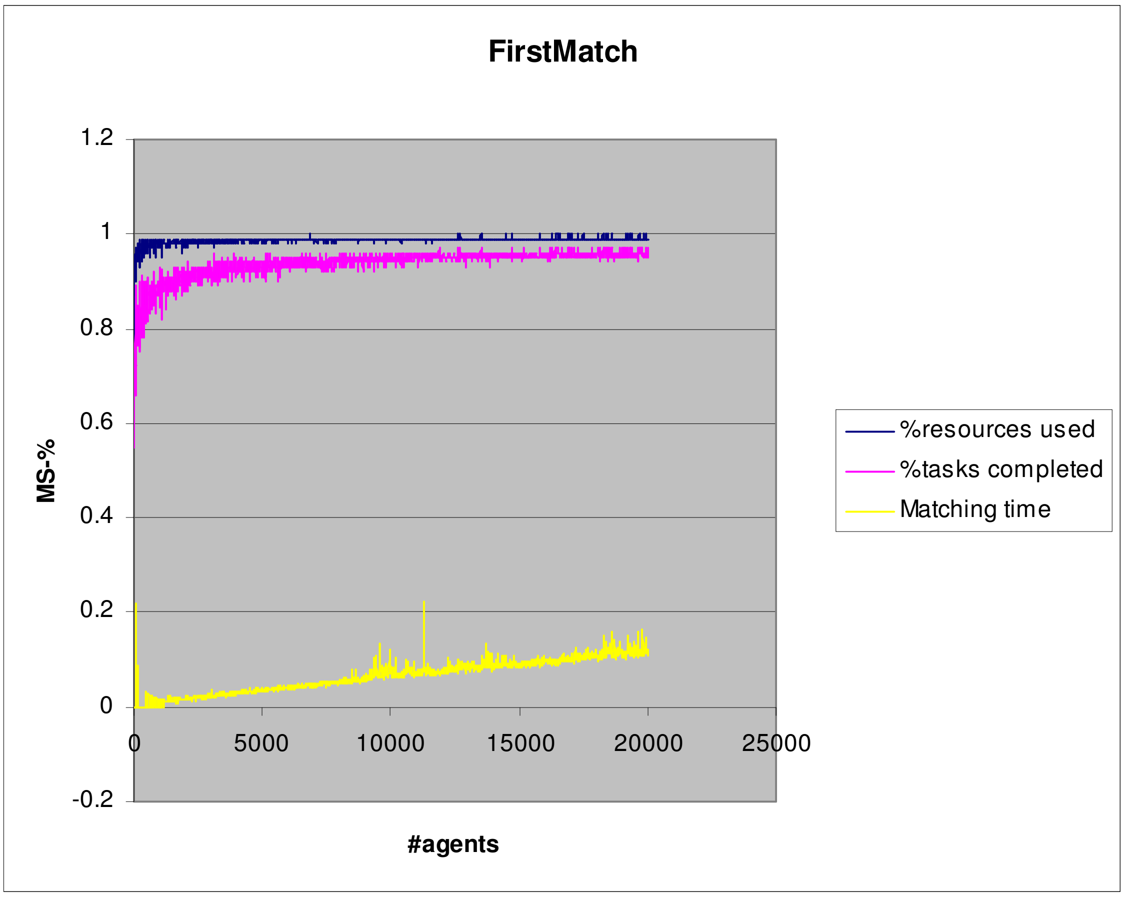


Figure1: Matching time and resource allocation for first match experiment taking into consideration 5-20000 agents.

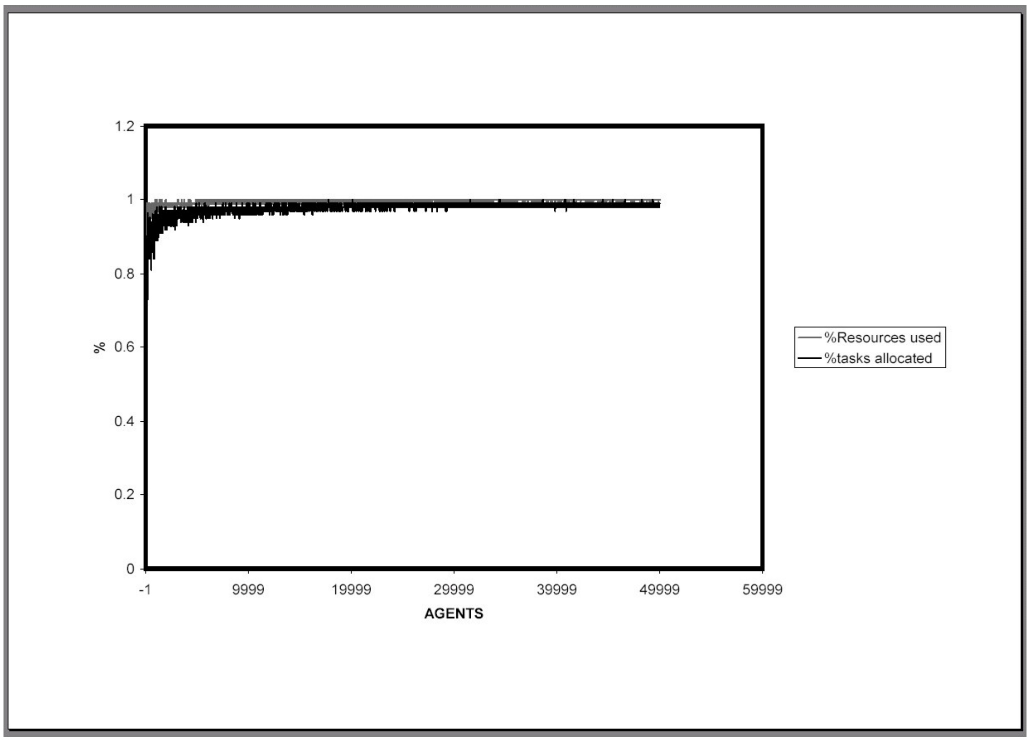


Figure2: Resource allocation for first match experiment taking into consideration 5-50000 agents.

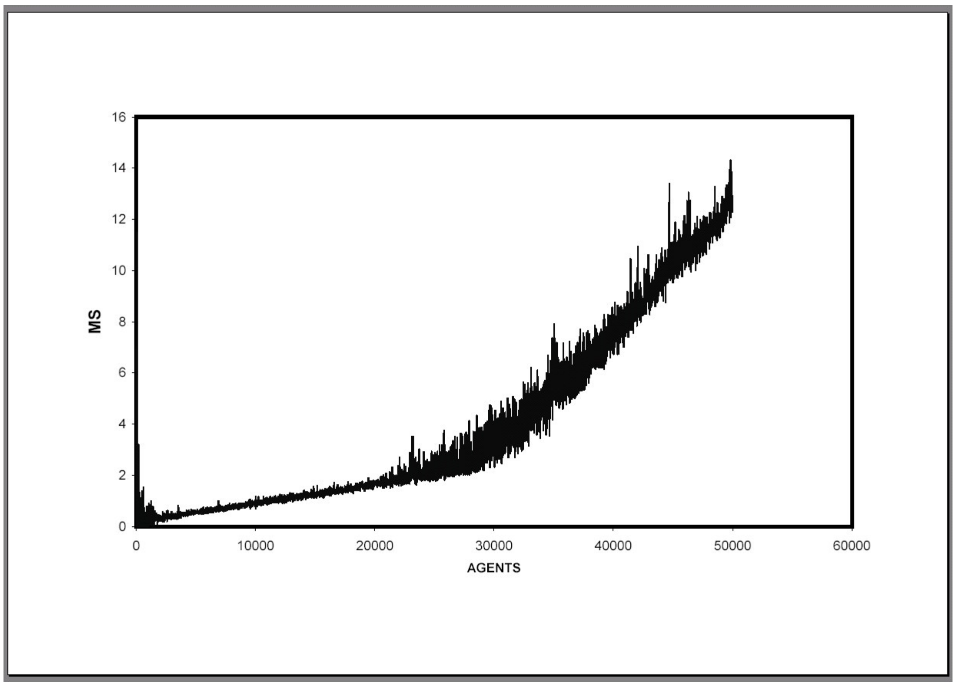


Figure 3: Matching time for first match experiment taking into consideration 5-50000 agents.

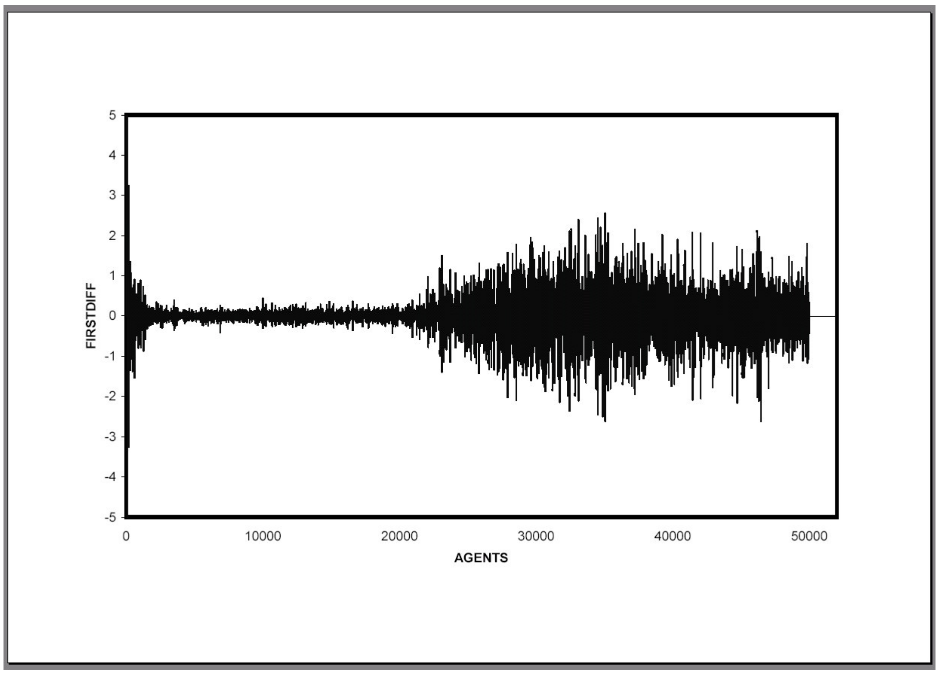


Figure 4: First difference of FirstMatch matching time.

4.2 Minimum difference:

What we wanted to investigate:

Task execution efficiency:
Percentage of allocated tasks of the different workloads.

Resource usage efficiency:
Percentage of available resources that are used up by the consumers.

Matching time.

Set up of experiment:

We start performing the simulation starting at 5 agents and move towards the 20000 range. The number of agents increases by a factor of 10 with each run. Values assigned for resources and tasks.These values are generated randomly between an interval of [0. . 10],[0. . 100] respectively.

Results:

In this case there is substantial time required to produce a match. This can be seen from figure 8.This is because for each consumer task that needs to be done we need to pick up a producer such that the difference between them is minimum. Even in this case the match time is proportional to the size and stability is observed on looking into first difference graph(figure 8).We can make a similar observation as for FirstMatch. For populations smaller than 5K, the matching is not very efficient with allocation percentages ranging around 90%. This allocation percentage rises to reach 96% for a 20K population size. Increasing the population from 15K to 20K, only increases the efficiency by 1% but requiring 50% more time to produce the match. Because of the very low improvement for either task or resources, we do not simulate beyond the 20K boundary. Shown in the same figure, we also plotted its first difference. From that graph, we can again observe that as the population size increases, the variance of the matching time goes up, introducing more uncertainty in the matching process. As far as the resource usage is concerned, and also shown in Figure 5,the same conclusion as for FirstMatch holds, it improves as population size increases and then saturates at 99%.

Graphs:

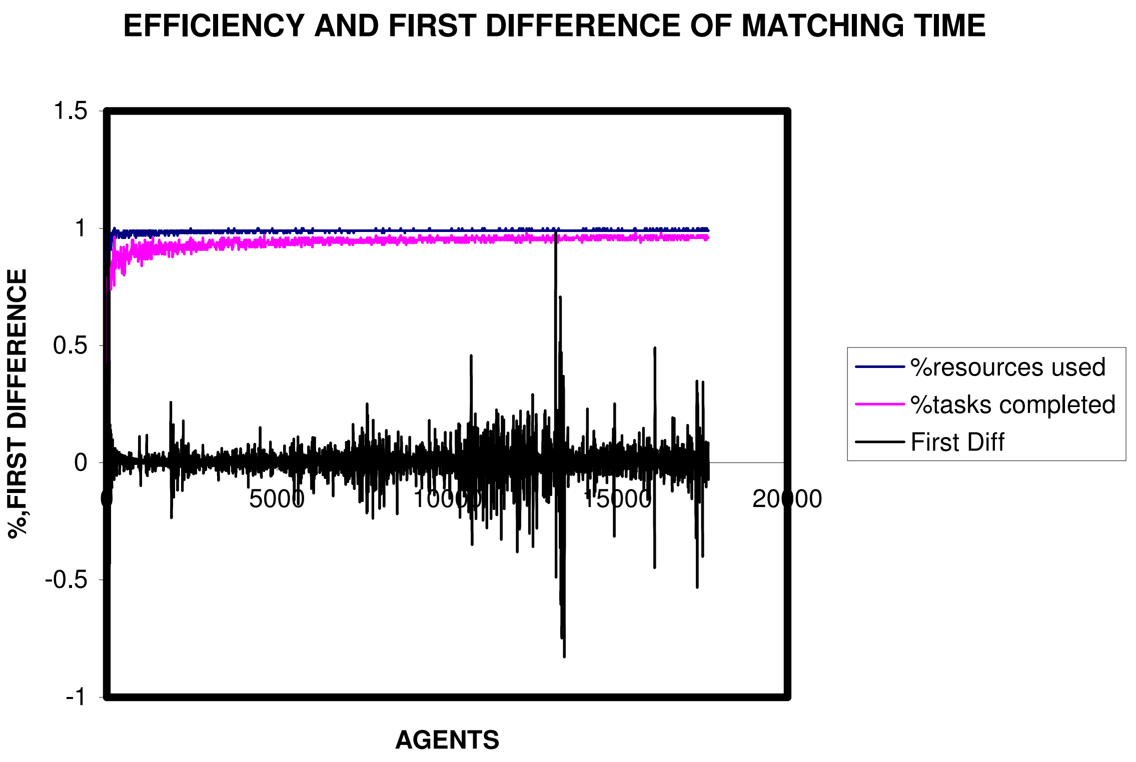


Figure 5: Minimum difference efficiency and first difference of matching time

4.3 Minimum distance:

What we want to investigate:

Task execution efficiency:
Percentage of allocated tasks of the different workloads.

Resource usage efficiency:
Percentage of available resources that are used up by the consumers.

Matching time.

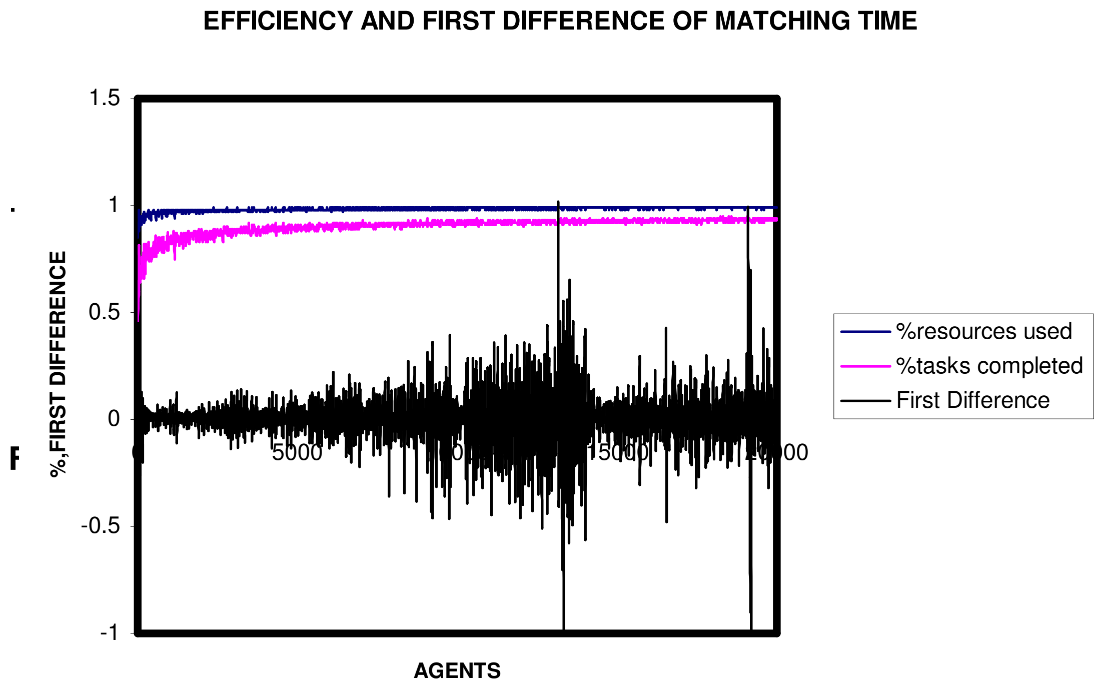
Set up of experiment:

We start performing the simulation starting at 5 agents and move towards the 20000 range. The number of agents increases by a factor of 10 with each run. Values assigned for resources and tasks. These values are generated randomly between an interval of [0. . 10],[0. . 100] respectively.

Results:

The last matching function that was used is the minimal distance between two agents. The minimum distance function computes the eucledian distance between each consumer and a given set of producers which is more expensive in terms of computing cycles than MinDifference and Firstmatch. We can clearly observe from figure 8 that the matching time of MinDifference is less than MinDistance. This is a direct consequence of the structure of the MinDistance function. Looking at the MinDistance efficiency plotted in Figure 7, a similar observation as for the other 2 functions can be made. The task execution efficiency does not increase substantially as we approach the 20k boundary. The first difference plot shows as similar behavior as the MinDifference. As far as the resource utilization is concerned, the same observations as above hold.

Graphs:


EFFICIENCY AND FIRST DIFFERENCE OF MATCHING TIME
1.5
1
0.5
0
-0.5
-1
%,FIRST DIFFERENCE
5000
10000
15000
20000
AGENTS
%resources used
%tasks completed
First Difference

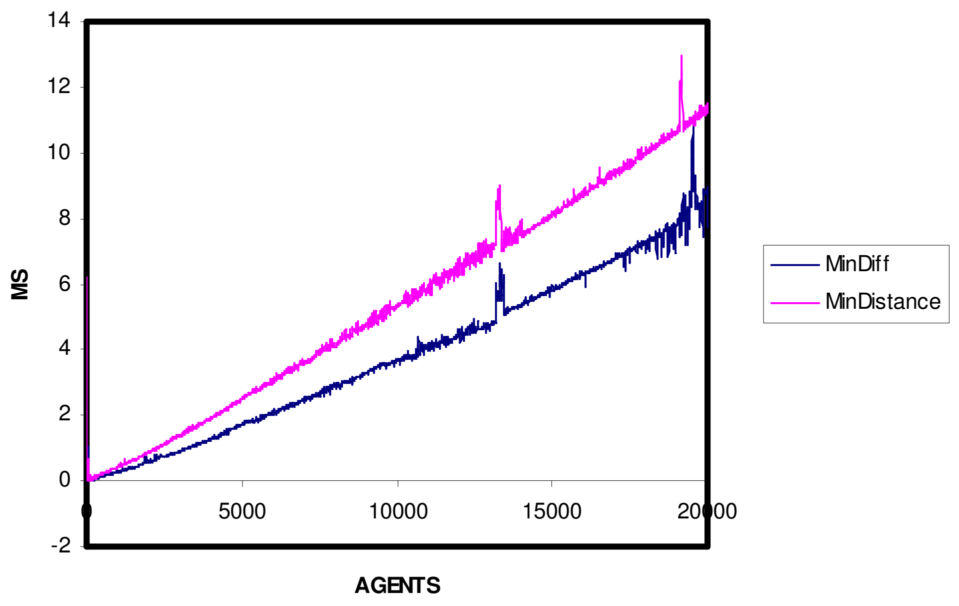

14
12
10
8
6
4
2
0
-2
MS
0
5000
10000
15000
20000
AGENTS
MinDiff
MinDistance

Figure 8: Comparisons in matching time between minimum difference and minimum distance functions.

4.4 Comparison between all the 3 functions:

Putting the results together and comparing the task allocation efficiency for the 3 functions, as plotted in Figure 7, we observe that MinDifference is the most efficient approach based on resource allocation. First match is the fastest function among them all taking matching time into consideration. MinDistance is slowest and also least efficient. We are looking for a function that is not only fast but also efficient. First match is simple and fast but not very efficient. On the other hand minDifference shows a trade off between efficiency and match time. Though it is slower than First match, it is the most efficient of them all.

Graphs:

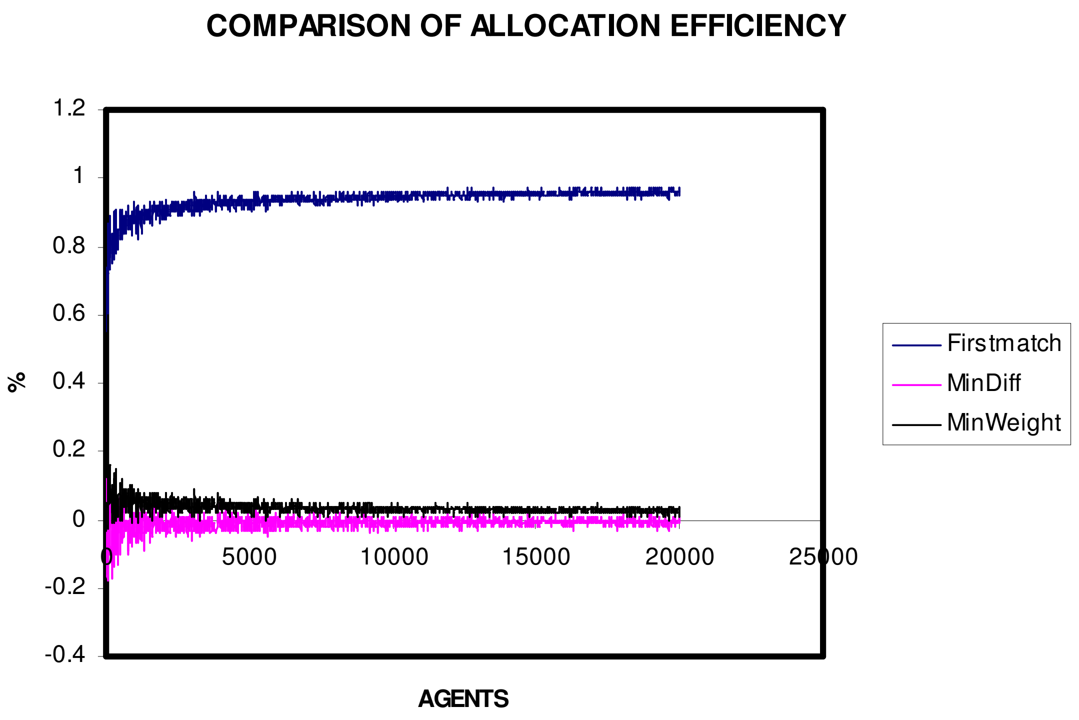


Figure 7: Allocation efficiency for the 3 functions

5 Extension of the experiment

We had used the centralized approach and used all the three functions that were built for matchmaking. As an extension to this experiment we applied partitioning techniques and then used each of these functions in the respective partitions.

5.1 Why partitioning?

In the centralized approach from the global view point all consumer agents and all producer agents participated in the match making process. Then we wanted to examine the behavior of these agents by having a local view. We restrict the scope of the producers and consumers that participate in the partitioning. One of the main criteria, which were used to restrict the view, was to build partitions and allow match making possible only between the consumer agents and producer agents that lie in the same partition. For e.g. if we have say partition 1 and partition 2. An agent in partition1 can match with another agent in partition 1 only and not in partition 2.

Plot of producer (X,Y) and consumer (X,Y)

The figures 9 and 10 indicate how uniformly the producers and consumers are placed in a centralized scenario before match making and partitioning is involved.

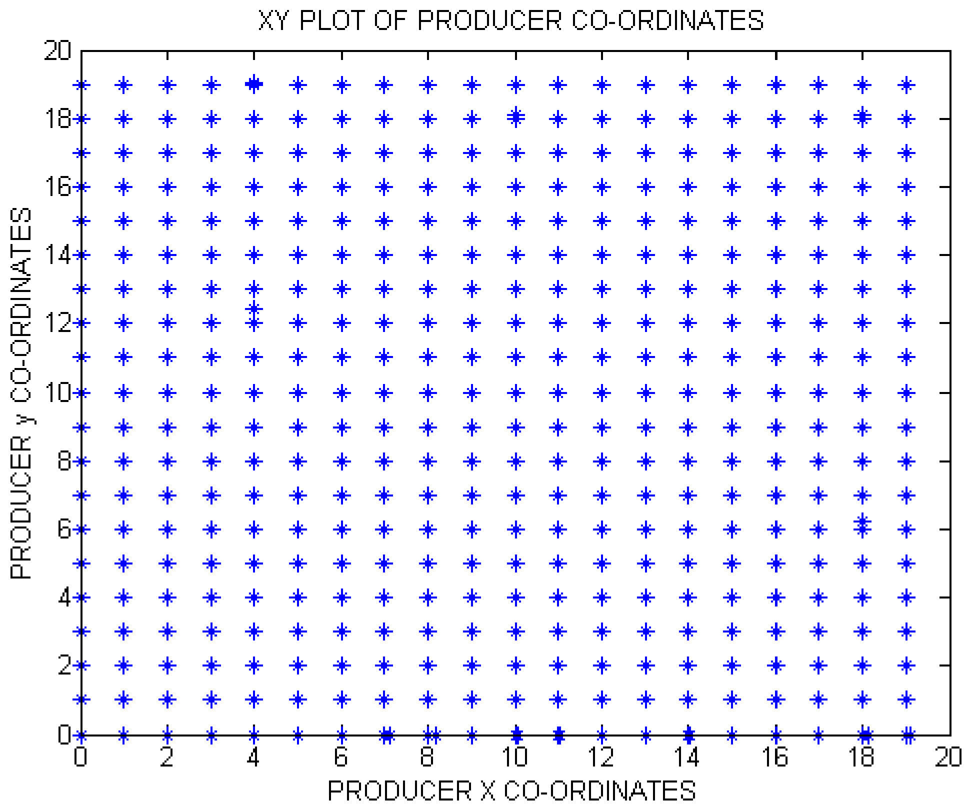


Figure 9: Diagram represents plot of producer co-ordinates

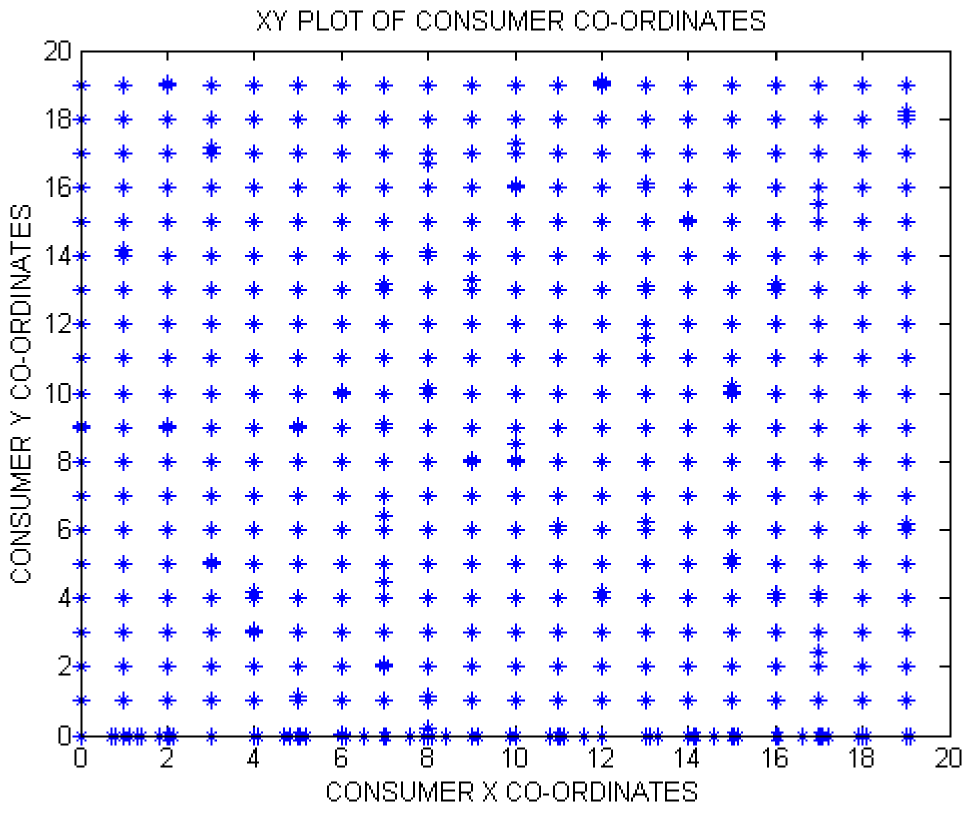


Figure 10: Diagram represents plot of consumer co-ordinates

5.2 Partitioning technique used:

We used a very simple partitioning technique based on the co-ordinates. Partitioning technique was based on cartesian co-ordinates. The co-ordinates were generated randomly. The number of partitions was taken as a user input. For our experiments we have used 4,8 and 16 partitions to examine the behavior. This can be extended to as many partitions as the user wants. A section of a given size is chosen and it is divided into many parts depending on how many partitions are needed. The floor operation was used so that points would lie in the particular partitioned section. Diagram 1 shows how this is done. Nx=2,Ny=2 indicate this scenario where a line Nx partitions x plane of diagram 1 into 2 equal halves and Ny partitions the y plane of the figure into 2 equal halves. Blue dots and yellow dots indicate the producers and consumers that lie in each of these partitions. The same approach is used for 8 partitions and 16 partitions. For e.g we can have 8 partitions using Nx=2,Ny=4 or Nx=4 ,Ny=2 etc. Many such combinations could be used to generate 8 partitions. But in our simulation we use Nx=4,Ny=2.Similarly for 16 partitions we use Nx=4,Ny=4.

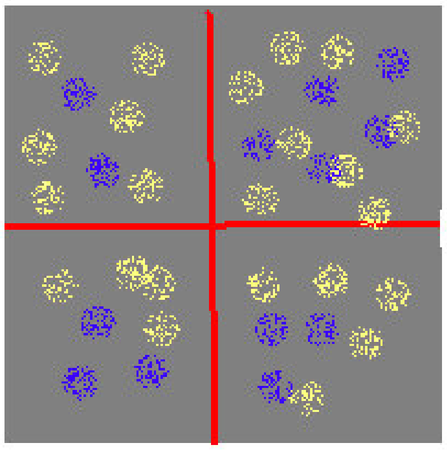


DIAGRAM 1 .THIS DEMONSTRATES HOW 4 PARTITIONS WITH PRODUCERS AND CONSUMERS LOOK LIKE.

5.3 Relation between this experiment and peer-peer scenario:

We wanted to move from the centralized scenario to peer-to-peer scenario. The partitioning based scenario gives us a view, which is close to the peer-to-peer scenario. Since our simulation, which is completely centralized, this wouldn't be strictly peer-to-peer. One of the major assumptions done was that there were be no message passing possible between agents. All our experiments are related to minimal agents. As mentioned before, in case of minimal agents, the constraint on the memory requirement is pretty high and the agents need to be as simple as possible. This was the main reason why we didn't build a protocol for exchange of messages. Making message passing possible can be a further extension to this experiment. This is left as future work.

5.4 Reasons for performing the experiment:

Our focus is on studying the behavior of matchmaking process on the local level in addition to what was already done on the global level. Some of the parameters, which would be seen to vary at the local level, are task execution efficiency, resource usage efficiency and matching time. There may be some partitions where there would be more consumer agents compared to producer agents and vice versa. The efficiency is such partitions would be less. This would give us idea about how these parameters are affected on partitioning.

5.5 Empirical Results:

We plotted the graphs for the matchmaking done using centralized approach and then using partitioning approach. As expected the partitioning based approach always results in lower efficiency when compared to the centralized approach. This is seen from figures 1,2 and 3.We have used 4,8,16 partitions in the extension to the basic experiment. One may be curious as to why the efficiencies of the original experiment and partitioning experiment seen in the minimum distance extension lie close to each other. The main reasons for this is that the minimum distance makes use of coordinate points in order to measure the distance. On partitioning the points lie very close to each other. This reduces the distance between the points in their respective partitions. This is the key factor, which influences the efficiency as a whole. This is observed from figures 7, 8 and 9.

5.6 First Match experiment extension:

Graphs:

We have figures (1,2,3) which are the graphs representing the first match experiment.
The number of agents over allocation efficiency is taken into consideration.

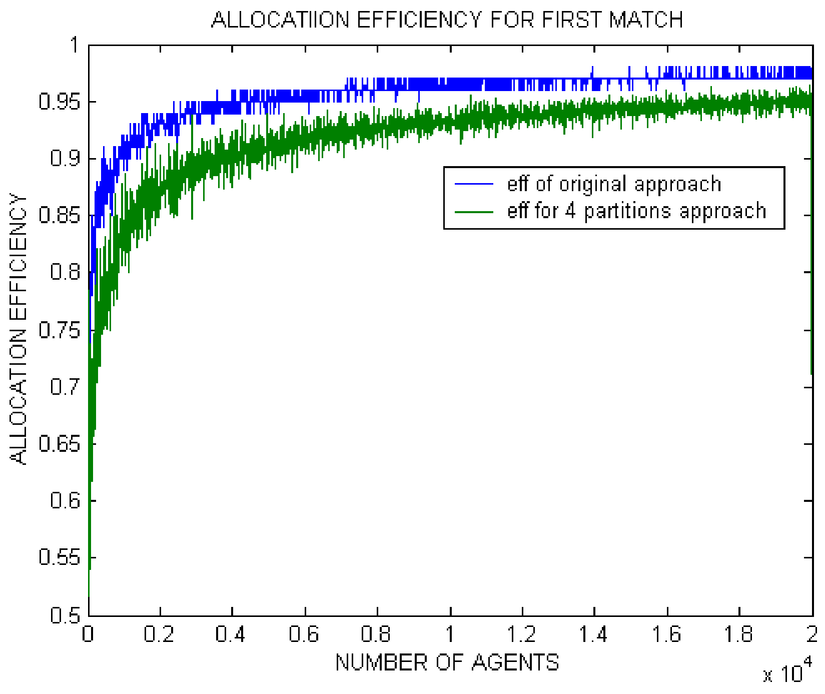


Figure 1: Allocation efficiency for first match using centralized approach and 4 partitions approach

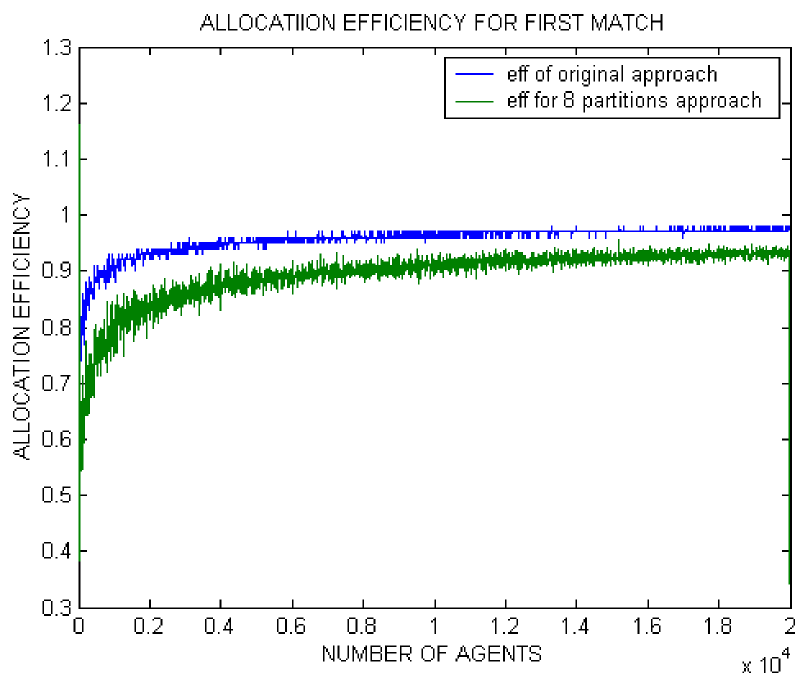


Figure 2: Allocation efficiency for first match using centralized approach and 8 partitions approach

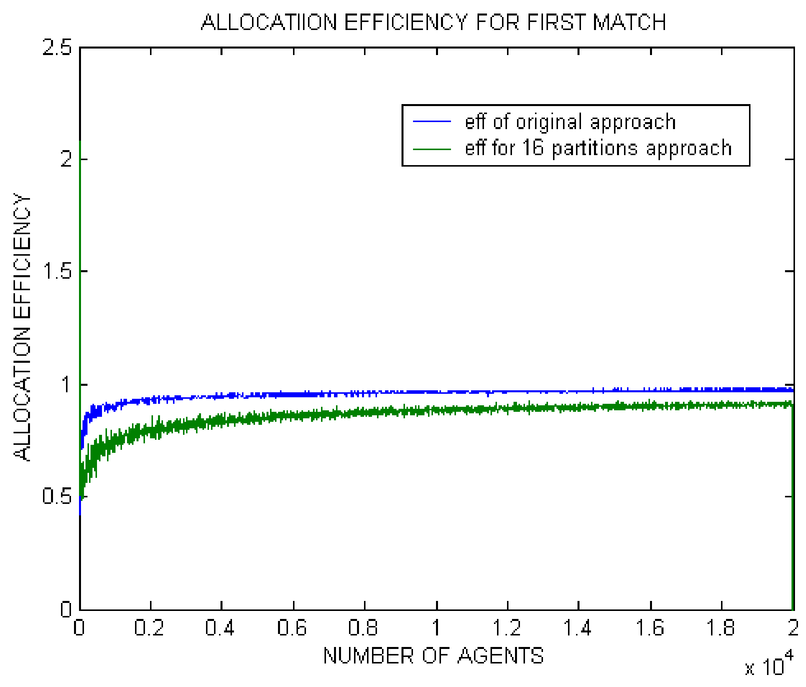


Figure 3: Allocation efficiency for first match using centralized approach and 16 partitions approach

5.7 Minimum Difference experiment extension:

We have figures (4,5,6) which are the graphs representing the minimum difference experiment. The number of agents over allocation efficiency is taken into consideration.

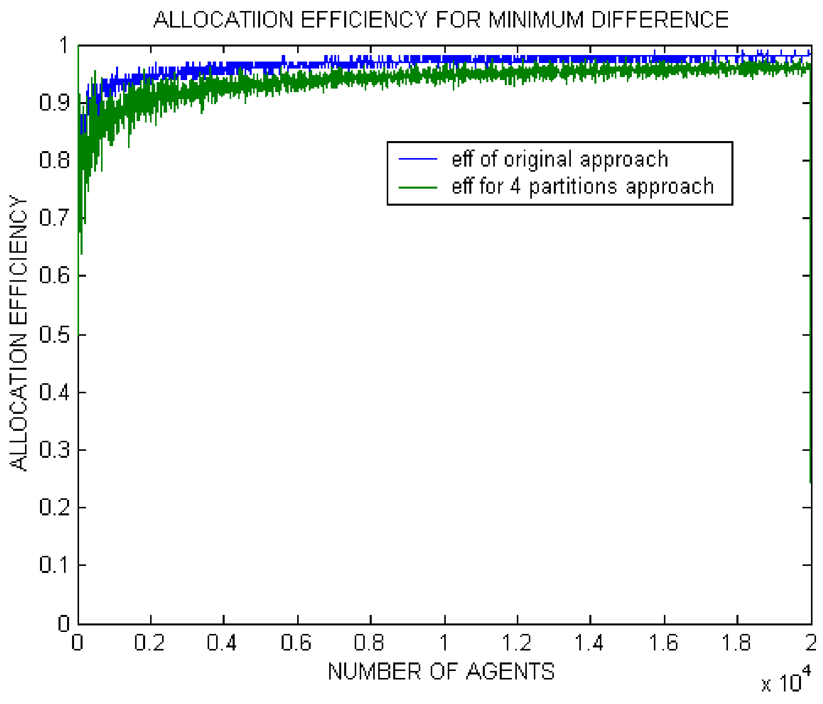


Figure 4: Allocation efficiency for minimum difference using centralized approach and 4 partitions approach

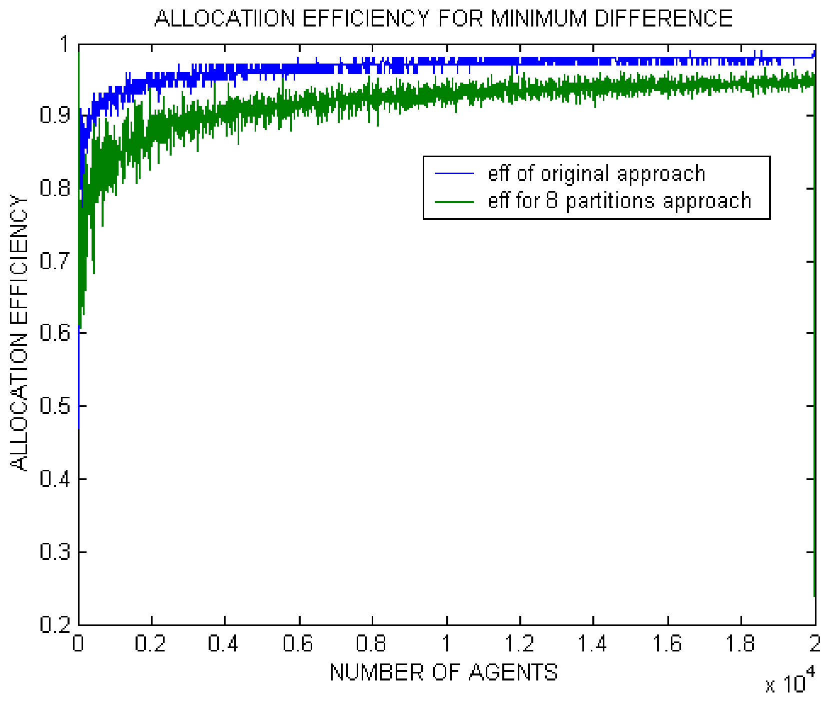


Figure 5: Allocation efficiency for minimum difference using centralized approach and 8 partitions approach

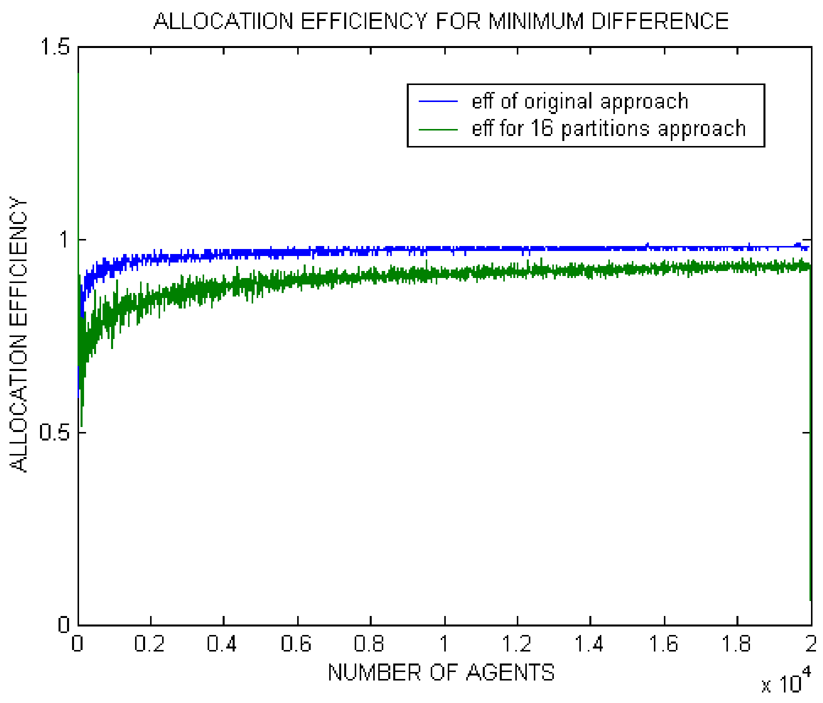


Figure 6: Allocation efficiency for minimum difference using centralized approach and 16 partitions approach

5.8 Minimum Distance experiment extension:

We have figures (7,8,9) which are the graphs representing the minimum difference experiment. The number of agents over allocation efficiency is taken into consideration.

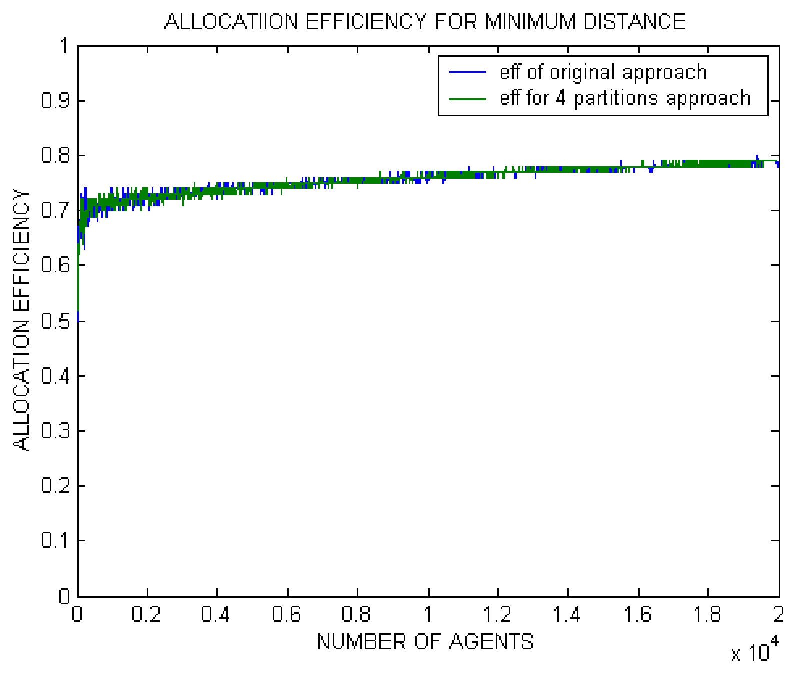


Figure 7: Allocation efficiency for minimum difference using centralized approach and 4 partitions approach

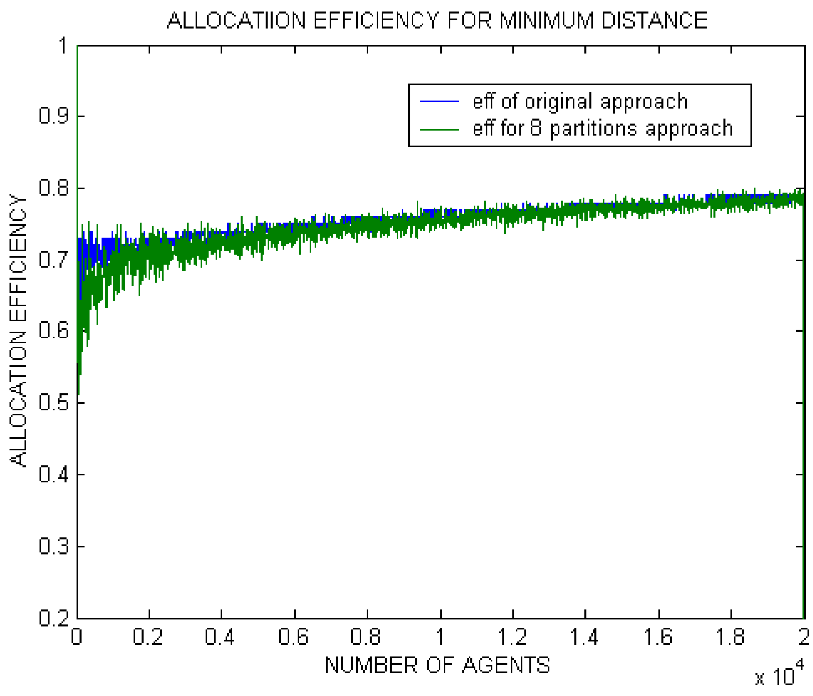


Figure 8: Allocation efficiency for minimum distance using centralized approach and 8 partitions approach

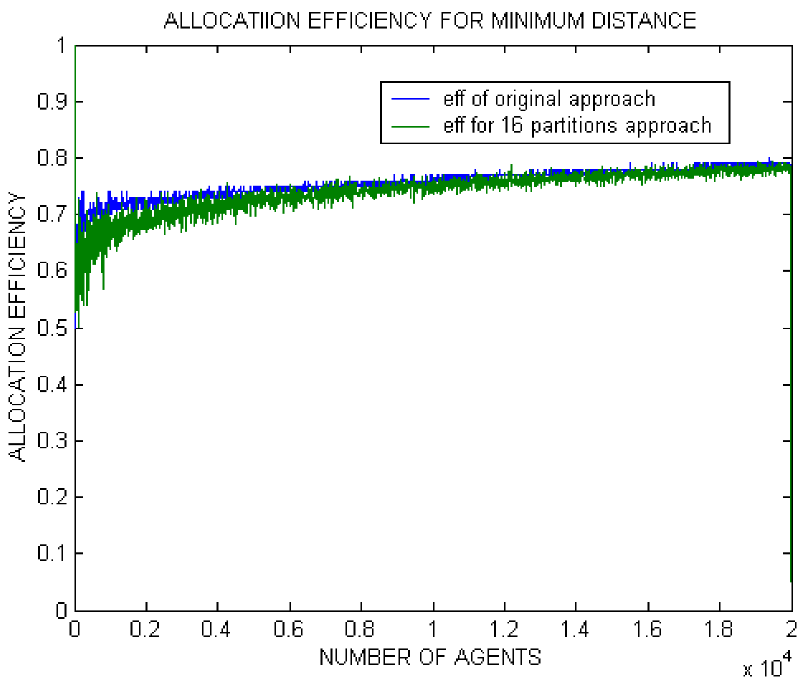


Figure 9: Allocation efficiency for minimum distance using centralized approach and 16 partitions approach

6 Commercial applications of matchmaking

There are several commercial applications for matchmaking experiments we performed. These simulations further strengthen the usage of such functions in real applications e.g. grid like environments, System on chip applications and in telecommunication networks. Matchmaking functions play an important role in matching resources in grid like environments. A grid consists of many interconnected systems with a few systems having sufficient resources and a few systems lacking such resources. Such simple matchmaking functions would be useful in overcoming complicated negotiations that would be needed during matchmaking. They are useful in network on chip applications where matching of memory and processes need to be done. This happens when there are some processes needing more memory and few processes having abundant memory in them. The memory can be suitable matched to the processes. The algorithms that are embedded in the chip perform such matching work. Other applications are seen in the area of telecommunication networks. Whenever resource allocation needs to be done in telecommunication networks simple matchmaking functions such as these are used. When we talk of resource it is bandwidth, it may always happen that some users may be using lot of bandwidth and other users may be starved of bandwidth. The network degrades due to this. Matchmaking aids in maintaining proper Quality of service(QOS) in telecommunication environments.

7 Conclusion and Future work

We have studied the resource allocation and matchmaking in single partition systems by giving a simple approach for matchmaking. We introduced a centralized matching mechanism that requires very little information to produce a good matching. In addition, the number of messages required to be broadcasted over the network is limited and in the order of O(N). When introducing multiple matchmakers, this can even be reduced to O(log N). We furthermore evaluated more complicated matching functions, as they incorporated more information for the actual matchmaking, by looking at task allocation efficiency, resource usage efficiency and matching time.

The main findings are:

- First-Match is the simplest and fastest in terms of matchmaking time.
- MinDifference is the most efficient of them all.
- Centralized matching mechanism easily scales up to 20K. For FirstMatch this even goes to 50K.
- It seems that there exists some kind of population size beyond or below which either no improvement can be generated or the matching efficiency goes down respectively. This allows the introduction of multiple matchmakers, reducing the number of messages to O(log N).

Issues which remain unsolved are how efficient this approach is when resources and tasks are unevenly distributed. It might be that more information intensive approaches will outperform these functions. It also remains to be seen if this approach remains feasible when introducing asynchronous, rather than batch-like, sequential, matchmaking. We further tried the clustering approach to see effect of clustering on matchmaking using this approach we always see lower efficiencies.

The main findings are:

- The efficiency plots for first match and minimum difference indicate that centralized approach always results in better efficiency compared to partitioning approach.
- The minimum difference approach is most suitable for clustered approach as the efficiency is almost the same as compared to partitioning approach.

The Partitioning based approach can be further extended to peer-to-peer approach, which is left for future work. We can have actual messages being sent between agents and this will help us determine which of the approaches the centralized approach or the peer to approach performs well under the given conditions.

8 References

[1]http://www-iiuf.unifr.ch/~brugger/papers/95_cidre/cidre/node26.html

[2]http://www.ai.sri.com/~oaa/main.html

[3] Kasselman C. Czajkowksi., Foster I. Resource Co-Allocation in Computational Grids. In proceedings of IEEE HDPC-8 August 1999.

[4] Epema D. Bucher A. Local versus global queues with processor co-allocation in multicluster systems. In *EighthWorkshop on Job Scheduling Strategies for Parallel Processing (in conjunction with HPDC-11)*, pages 184–204. 2002.

[5] Karonis N. Kesselman C Martin S. SmithW. Tuecke S. Czajkowski K., Foster I. A resource management architecture for metacomputing systems. In *The 4th workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82. 1998.

[6] Ch.Weinhardt D. Veit, J.P. Muller. Multidimensional matchmaking for electronic markets. *Journal of Applied Artificial Intelligence*, 16(9-10):853–869, 2002.

[7] Klusch M. Widoff S. K. Sycara, Lu J. Matchmaking among heterogeneous agents on the internet. In *Proceedings. AAAI Spring Symposium on Intelligent Agents in Cyberspace*. 1999.

[8] Harada L. Kuokka, D. Matchmaking for information agents. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 672–678, 1995.

[9] Jennings N. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.

[10] E. Ogston and S. Vassiliadis. Matchmaking among minimal agents without a facilitator. In *Proceedings. 5th International Conference on Autonomous Agents*, pages 608–615. May 2001.

[11] E. Ogston and S. Vassiliadis. A peer-to-peer agent auction. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems Part I*, pages 151–159, July 2002.

[12] E. Ogston and S. Vassiliadis. Unstructured agent matchmaking: experiments in timing and fuzzy matching. In *Proceedings of the 2002 ACM symposium on applied computing*, pages 300–306, March 2002.

[13] Milind Tambe Makoto Yokoo Pragnesh Jay Modi, Wei-Min Shen. Asynchronous complete method for general distributed constraint optimization. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems Part I*, July 2002.

[14] Jennings N.R. Vulkan, N. Efficient mechanisms for the supply of services in multi-agent environments. *Journal of Decision Support Systems*, 28(1-2):5–19, 2000.

Appendix

Pseudocode for all three match making functions

First Match

```
start :

outer : = minimum_num_of agents;

while (outer < maximum_num_of_agents) {

count: = outer;
 i: = 0;

  /*Reading elements into producer vector */

while (i < count) {
    read_values(producer);
    i: = i + 1;
  }

  /*Reading elements into consumer vector*/

while (i < count) {
    read_values(consumer);
    i: = i + 1;
  }

  /* Initialize the state */

  state: = -1;

 while (count > 0) {

    if (state = = -1) {
      c: = consumer_element_at_index(0);
      con_remove: = remove_consumer_element_at(0);
      index: = findBestProducer(c, producerinfo, count);
      count: = count - 1;
      state: = 2;
    }
if (index == sizeof(p)) {
      state= -1;
   }

else {
      state = 2;:
   }

if (state = = 2) {
      pro_remove: = remove_producer_element_at(index);
      print("Match found at p and con);
      state: = -1;
   }

  }

outer = outer + 1;

}

/*end of outer loop */

: end


 Procedure calls :

 /* This procedure returns an integer value  */

Integer : findBestProducer(Consumer con, Producer p, int count) {

count: = c;
 j: = 0;

 while (j < sizeof(p)) {

    if (doesMatch(pro, co)) {
      return j;
    }

  }

}
```

```
/* This procedure returns a boolean value */

boolean : doesMatch(Producer p, Consumer c) {
  if (c <= p) {
    return (true);
  }

else {
    return (false);
  }
}
```

Minimum Difference

```
start:

outer:=minimum_num_of agents;

while (outer < maximum_num_of_agents) {

count: = outer;
 i: = 0;

  /*Reading elements into producer vector*/

 while (i < count) {
    read_values(producer);
    i: = i + 1;
  }

  /*Reading elements into consumer vector*/

while (i < count) {
    read_values(consumer);
    i: = i + 1;
  }

  /* Initialize the state */

state: = -1;
while (count > 0) {

    if (state == -1) {
      c: = consumer_element_at_index(0);
      con_remove: = remove_consumer_element_at(0);
      index: = findBestProducer(c, p, count);
      count: = count - 1;
      state: = 2;
    }

    if (index == sizeof(p)) {
      state:= -1;:
    }

 else {
    state:= 2;:
    }


if (state == 2) {
      pro_remove: = remove_producer_element_at(index);

      state: = -1;
    }

  }

  outer: = outer + 1;

}
/*end of outer loop */

: end
```

Procedure calls :

```
/* This procedure returns an integer value  */

Integer : findBestProducer(Consumer con, Producer p, int count) {

count: = c;
 j: = 0;
while (j < sizeof(p)) {

    if (j == doesMatch(pro, co)) {
      return j;
    }

  }

}



/* This procedure returns a boolean value */

double : doesMatch(Producer p, Consumer c) {
  if (c <= p) {
    return (value(p) - value(c));
  }
  else {
    return ( -1);
  }
}
```

Minimum distance

```
start:

outer:=minimum_num_of agents;

while (outer < maximum_num_of_agents) {

count: = outer;
 i: = 0;


  /*Reading elements into producer vector */

while (i < count) {
    read_values(producer);
    i: = i + 1;
  }
  /*Reading elements into consumer vector*/

while (i < count) {
    read_values(consumer);
    i: = i + 1;
  }

  /* Initialize the state */

state: = -1;


while (count > 0) {

    if (state == -1) {
      c: = consumer_element_at_index(0);
      con_remove: = remove_consumer_element_at(0);
      index: = findBestProducer(c, p, count);
      count: = count - 1;
      state: = 2;
    }

    if (index == sizeof(p)) {
      state: = -1;
        }

    else {
      state: = 2;
    }


if (state == 2) {
      p: = producer_element_at_index(index);
      pro_remove: = remove_producer_element_at(index);
      print("Match found at p and con);
      state: = -1;
    }

  }



  outer: = outer + 1;

}
/*end of outer loop */

: end
```

Procedure calls :

```
 /* This procedure returns an integer value  */

Integer : findBestProducer(Consumer con, Producer p, int count) {

count: = c;
j: = 0;

while (j < sizeof(p)) {

if (doesMatch(pro, co)) {
      return j;
    }

  }

}



/* This procedure returns a double value  */

double : doesMatch(Producer p, Consumer c) {

 if (c <= p) {
    diff_x_value: = pro_x_coordinate - con_x_coordinate;
    diff_x_value: = diff_x_value ^ 2;

    diff_y_value: = pro_y_coordinate - con_y_coordinate;
    diff_y_value: = diff_x_value ^ 2;

    diff_total_value: = sqrt(diff_x_value + diff_y_value);
    return diff_total_value;
  }

 else {
    return -1;
  }

}
```