A Fuzzy-Logic Based Coordinated Scheduling Technique for Inter-grid Architectures

Abdulrahman Azab, Hein Meling, Reggie Davidrajuh

A Fuzzy-Logic Based Coordinated Scheduling Technique for Inter-Grid Architectures

Abdulrahman Azab[1,2] Hein Meling[1] Reggie Davidrajuh[1]

[1] Dept. of Electrical Engineering and Computer Science, Faculty of Science and Technology, University of Stavanger, Norway
abdulrahman.azab@ux.uis.no,{hein.meling,reggie.davidrajuh}@uis.no
[2] Dept. of Computer and Systems Engineering, Faculty of Engineering Mansoura University, Egypt
abdulrahman.azab@mans.edu.eg

Abstract. Inter-grid is a composition of small interconnected grid domains; each has its own local broker. The main challenge is to devise appropriate job scheduling policies that can satisfy goals such as global load balancing together with maintaining the local policies of the different domains. Existing inter-grid methodologies are based on either centralised meta-scheduling or decentralised scheduling which carried is out by local brokers, but without proper coordination. Both are suitable interconnecting grid domains, but breaks down when the number of domains become large. Earlier we proposed SLICK, a scalable resource discovery and job scheduling technique for broker based interconnected grid domains, where inter-grid scheduling decisions are handled by gateway schedulers installed on the local brokers. This paper presents a decentralised scheduling technique for the SLICK architecture, where cross-grid scheduling decisions are made using a fuzzy-logic based algorithm. The proposed technique is tested through simulating its implementation on 512 interconnected Condor pools. Compared to existing techniques, our results show that the proposed technique is better at maintaining the overall throughput and load balancing with increasing number of interconnected grids.

1 Introduction

Grid computing provides the infrastructure for aggregating different types of resources (e.g. desktops, mainframes, storage servers) for solving intensive problems in different scientific and industrial fields, e.g. DNA analysis, weather forecasting, modelling and simulation of geological phenomenon [1]. Based on the delivered service, grid computing can be classified into computational grids, and data grids [2]. The target of computational grid, which is our main focus, is to aggregate many grid compute resources as one powerful unit on which computational intensive applications can run and produce results with low latency. Computational grid model is mainly composed of three components: (i) clients that consume grid resources by submitting computational jobs, (ii) resource brokers who are responsible of allocating submitted jobs to matching workers, and

(iii) workers that execute submitted jobs. Due to the rapid growth in the demand for compute resources, as more and more compute intensive applications are being built, there is a need to scale up the grid infrastructure with more workers to fulfil those demands. To scale up, without negatively impacting overall performance and throughput is challenging, as increasing the number of nodes in a single grid could lead to a performance bottleneck since the broker may become overloaded.

Introducing multiple load balanced brokers for the same resource set requires some form of coordination between the brokers [3]. One possible solution is to establish an interconnection between existing grid domains. This would enable migration of grid jobs between domains to avoid job starvation. The concept of interconnecting grid domains was first introduced by the Condor project in 1992, by means of a technique called *flocking* [4]. The initial idea was to use *gateway* machine in each Condor pool, i.e. a grid domain, to manage the interconnection between domains. Due to the lack of coordination between the broker and the gateway inside domains, this has now been replaced by a client initiated approach [5], which enables client nodes to submit jobs to external brokers when no matches are found by their local brokers. This however, is not practical when a large number of interconnected domains must be configured, each with hundreds of clients that communicate with many external brokers. Other grid systems (e.g. gLite [6], Condor-G [7], UNICORE [8], Nimrod-G [9]) used the concept of meta-scheduling [10], known also as super-scheduling. Meta-schedulers work in a layer above traditional brokers so that instead of submitting to their local brokers, users submit their demands to a meta-scheduler which transfers the submissions to a broker on a grid domain which can fulfil the demand. The problem with this approach is the lack of coordination between meta-schedulers. Another approach, *broker overlay*, is to establish the interconnection through an overlay network between brokers [11–15, 3]. This approach has proven to be scalable [11] but it doesn't achieve load balancing. The reason is that for each external job submission, the local broker sends a query to its neighbouring brokers looking for a match. Any matching outside its neighbourhood will not be detected. Avoiding this problem by constructing a fully-connected logical topology between brokers is not practical, since that will result in increased scheduling overhead as the number of links between brokers increase.

This paper presents a coordinated scheduling technique for interconnected grids, which is an addition to our Slick broker overlay architecture [16]. In Slick, brokers maintains resource information about workers in its domain in a reduced information set (RIS) data structure. The RIS of each domain is disseminated to other brokers through a simple gossip based protocol [17]. Thus, each broker obtains a RIS from all the other domains in the system, and maintains this information in a data structure that we call global information set (GIS). Jobs with no matching workers in their local domains are exported to external domains with matching workers. Selecting the domain to export a job to is performed through a matchmaking process between the job requirements and the resource attributes of each RIS in the local GIS. Previously in Slick [16], we

proposed a simple matchmaking technique which finds any match. This paper proposes a fuzzy scheduler that finds the best matching domain for jobs.

The proposed technique is modelled using the PeerSim P2P network simulator [18]. Using the simulation model we have tested the proposed technique by interconnecting simulated Condor pools [19] using SLICK. Job allocation throughput and load balancing of the proposed technique is tested against existing scheduling techniques. The results show that for a system capacity of 50,000 nodes divided into 512 domains, the proposed technique achieves load balancing and reasonable throughput. In the broker overlay, we used a broker-to-broker neighbourhood degree of $k = \frac{\ln N}{\ln 2}$, where N is the total number of brokers.

The paper is organised as follows: Section 2 gives the background for the problem, and surveys related work. Section 3 describes the design of the SLICK gateway and the Fuzzy scheduling technique. Section 5 presents the simulation model and our evaluation. Section 6 concludes the paper.

2 Scheduling in Inter-Grid Architectures

The interconnection of grid domains may be implemented in one of three levels: (1) *Client level* where the client machine can have access to multiple domains using associated access rights [4, 20], (2) *Worker level* where worker nodes can install the task executors of multiple domains and thus becomes available for task submissions from either of those domains [21, 6], and (3) *Broker level* where the interconnection is carried out through local resource brokers. Two different methodologies follow the latter approach: (a) *Central meta-scheduler*, and (b) *broker overlay*. The role of a central meta-scheduler [10] is to manage the inter-grid submission requests, allocating each to a broker with matching resource requirements in its domain [7, 9]. Scheduling in broker overlay architectures can be further classified into: (i) *non-coordinated*, and (ii) *coordinated* [3]. For non-coordinated scheduling, there is no cooperation between brokers making scheduling decisions for tasks assigned to external domains. One methodology is to have a shared directory service, as in UNICORE [8], where brokers can retrieve information about workers in external domains for making cross-domain scheduling. Another methodology is implemented in Tycoon [15], where each broker declares a public independent auction for its available resources and external brokers can compete to gain access to them. For coordinated scheduling, a broker communicates with other brokers for negotiation about their available resources before taking the inter-grid scheduling decision. This communication is implemented through two main methodologies: (A) *Broadcasting*, e.g. one-to-all, where cross-domain job requests are sent to all external brokers and take decisions based on positive responses [13, 12, 14]. This methodology is applicable only for small number of domains, as otherwise the negotiation process would negatively influence the overall throughput. (B) *Multi-casting*, where instead of negotiating with all external brokers, each broker only negotiate with a select set of neighbouring brokers based on a broker overlay structure [3, 11, 12, 14]. This methodology is both scalable and fault-tolerant, but due to the need to disseminate of resource information, its ability to identify available resources depends

on the neighbourhood degree [22]. This means that for a broker to search for some specific resource information, multi-hop communication may be necessary, since a broker overlay network can offer reliable dissemination among available brokers. However, there can be a significant latency involved in such multi-hop communication [23, 22]. Due to this obstacle, coordinated-scheduling based inter-grid systems only rely on a small set of resource specification parameters, e.g. three parameters [3], for coordination.

In SLICK we implement multi-casting based coordinated scheduling, where each broker keeps a replica of the resource information of each external domain, and periodically synchronises local replicas with its neighbours.

3 Design

SLICK performs inter-grid job scheduling over a broker overlay of interconnected grid domains. Each broker in the overlay has a fixed number of neighbours $k < N$, where N is the total number of brokers. The k value is determined by the overlay topology, that is, $k = 2$ for the Ring topology, and $k = \frac{\ln N}{\ln 2}$ for the Hypercube topology. The system architecture is shown in Figure 1.

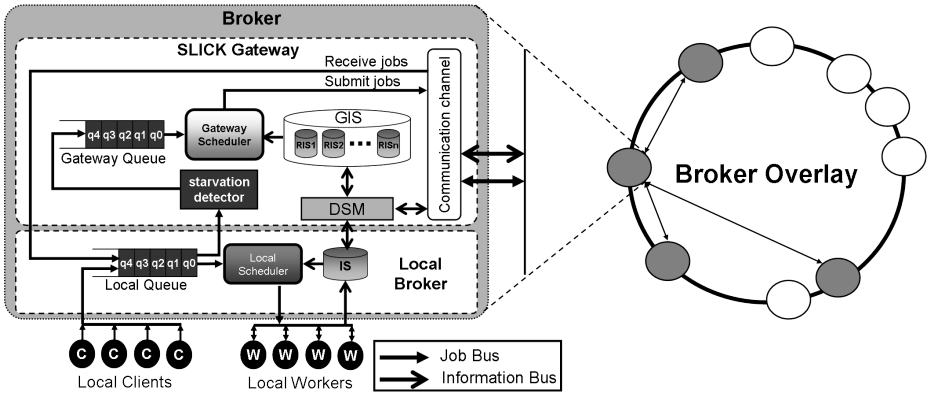


Fig. 1: System Architecture

The SLICK *gateway* is designed to be installed on broker nodes as an additional layer on the top of the broker to manage the interaction with external domains. In this paper, we present our implementation of the SLICK gateway for Condor[3] [19] interconnected domains.

3.1 Condor

Condor provides a way to harness idle computational cycles for use by computational intensive distributed applications. A Condor pool is composed of: (i) *users* (clients), which submit jobs for execution, (ii) *executors* (workers), which runs

[3] The new name of the Condor project is 'HTCondor'.

compute jobs, and (iii) a *central manager* (broker), which is responsible for collecting job submission requests and allocate jobs to workers. One machine can operate as both client and worker. Condor uses an extensive machine and job description language called ClassAds [24]. The ClassAd describing a machine's computational power is simply number of *slots* that it can support, which is equal to the number of CPU cores on the machine. A Condor ClassAd job description can represent: (i) a *Simple job*, which has one process and needs one slot to run or (ii) a *Job cluster*, which is a collection of simple jobs and needs n slots to run, where n is the number of jobs in the job cluster. Upon a job submission to the broker, it is placed in a queue. The job waits until the broker finds a worker with resources matching the job requirements. Matchmaking is carried out between the job's ClassAd and each worker's machine ClassAd. The job is allocated to the first matching worker with idle slots. Jobs of one job cluster can be allocated to multiple worker machines based on which matching workers have idle slots.

3.2 Domain Resource Description

Condor worker software automatically detects the number of CPU cores and the size of the virtual memory of the local machine, in addition to other machine specifications (e.g. operating system), and builds the machine ClassAd. Each machine ClassAd contains more than 100 attribute-value pairs, which would result in a large amounts of network traffic in the broker overlay, if we want to disseminate the machine ClassAds of the entire domain to other domains. To overcome this problem, we divide machine ClassAds into groups based on similarities in the hardware and software specifications, e.g. all Linux machines with x86_64 architecture, physical memory > 2 GB, supports Java, ..., are in group 1. The number of machine group is the number of all available machine specifications which is supported by Condor [25], and exist in $\geq$ 1 worker machines in the system. The domain resource description is stored in the SLICK gateway as a RIS, which is composed of: (i) a collection of key-value pairs, where the key is a machine group id and the value is the total number of slots which machines belong to the associated group. (ii) a time-stamp, ts, representing the read time of the RIS. The structure of the RIS is described in [16].

3.3 Resource Information Sharing

As described above, in addition to its local RIS, each domain keeps a replica of the RIS of the other domains in the system in the GIS, as shown in Figure 1. The RIS replication between brokers in the overlay is carried out by a data synchronisation manager, DSM. The DSM in each broker implements a simple anti-entropy gossip protocol [17], to synchronise the local GIS with its neighbours. The gossip protocol is described in [16].

4 Inter-Grid Scheduling

The roles of the SLICK gateway are: (i) Receive jobs submitted by external brokers throughout the overlay and schedule them to the local job queue[4]. (ii) Sched-

[4] This may result in contention between local jobs and external jobs. We are planning to add a contention manager to avoid this problem in the future.

ule starving local jobs to external brokers. A job j is declared as *starving* by the *starvation detector* (see Figure 1), if (CurrentTime - SubmissionTime(j)) $>$ β, where β is a timeout value. Whenever a job is detected to be starving, it is moved to the gateway queue (GQ). The gateway scheduler performs matchmaking between jobs in GQ and each RIS stored in the GIS, and submits each job to a matching broker.

In our previous paper introducing the SLICK [16] architecture, we implemented a simple matchmaking technique, *any-match*. Matchmaking was performed between the job description ClassAd and the RIS replicas of only neighbour domains in the overlay. Upon finding any match D_x for job j, implied that domain D_x contained one or more workers who's machine group matches the requirements of job j. This would result in job j being allocated to domain D_x. This technique works well for scheduling simple jobs. In this section we introduce a new matchmaking technique, *best-match*, where either a simple job or a job cluster is allocated to the domain with the highest match. The new technique is illustrated with an example of a job cluster in Figure 2.

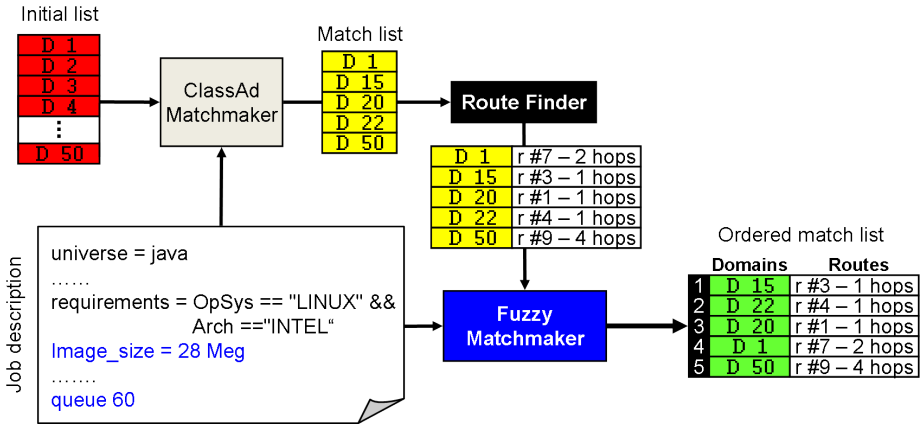


Fig. 2: Matchmaking of a Condor job cluster using the SLICK gateway scheduler.

1. An initial matchmaking is performed between the description of the job cluster j and each $RIS_x \in GIS$ of a domain D_x. The ClassAd matchmaker works similar to the Condor matchmaker, called *condor_negotiator*, where the result is a true/false value for each RIS. The result is true for D_x if there exists a set of workers $\{W : |W| > 1 \wedge W \in D_x\}$, and the machine group of each worker $w \in W$ matches the job requirements of j. In the example above, `Arch` is Intel, `OpSys` is Linux, and Java is installed. The output is a list of matching domains, *match list*.
2. The match list is passed to the *router*, which finds the route from the local domain D to each domain D_x in the list. If the brokers of D and D_x are neighbours in the overlay, then the route is a direct path with zero hops.

Otherwise, the route with the smallest number of hops between D and D_x, $r(D_x)$ selected. The route description is added to each domain in the list.

3. The new match list together with the job description are passed to the fuzzy matchmaker, where an *ordered match list* is generated based on best-match-first. The role of the fuzzy matchmaker is described in Section 4.1.
4. The local gateway scheduler allocates j to the first domain in the list. The other domains in the list are substitutes in case of failure or rejection.

4.1 Fuzzy Matchmaker

The fuzzy matchmaking is implemented based on Takagi-Sugeno fuzzy model [26], in which output membership functions are either linear or constant. In this implementation, constant output membership functions are used. Here we continue using the example in Figure 2, but with only three domains in the match list for simplicity. The following steps are carried out by the fuzzy matchmaker:

Generation of Input Membership Functions Each domain in the match list is viewed as a fuzzy set. Each fuzzy set has two *dynamic* input membership functions, related to two crisp inputs: (i) The number of CPU slots required by the job (60 slots in Figure 2), and (ii) Image_size which is the maximum amount of memory consumed by the job while running on the worker node (28 MB in Figure 2). The input membership functions for each domain are generated as follows:

1. The fuzzy matchmaker contacts the broker of each domain D_x in the list to obtain two values: (i) the current number of idle slots v_{D_x}, and (ii) the current queue size q_{D_x}.
2. The ♯ CPU slots membership function is generated by (1), where S_{D_x} is the set of all CPU slots in D_x, read from RIS_x, and v_{D_x}. Three examples of ♯ CPU slots membership functions are shown in Figure 3a, where $|S_{D_1}| = 100$, and $v_{D_1} = 35$

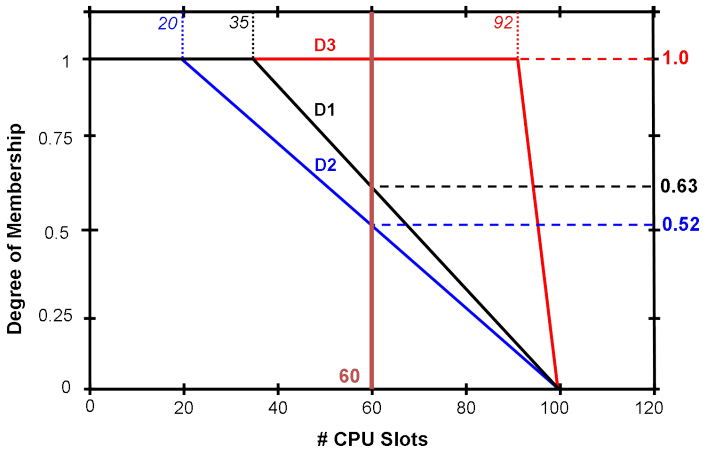
$$Y_{D_x}(x) = \begin{cases} 1 & x \in [0, v_{D_x}] \\ \frac{x - |S_{D_x}|}{v_{D_x} - |S_{D_x}|} & x \in [v_{D_x}, |S_{D_x}|] \end{cases} \quad (1)$$

3. The Image_size membership function is generated by (2) based on the fact that a job with an image size I must be allocated to a worker with virtual memory capacity $\geq I$. The x-axis is divided into intervals $(x_{i-1}, x_i]$, where $i \in \{1, 2, \ldots, 6\}$, assuming that the image size of any job won't exceed 60 MB. The Image_size membership function of domain D_x is generated as follows: For each interval $(x_{i-1}, x_i]$, we generate a set of CPU slots S_{x_i} as a subset of the total CPU slots in the domain, and the virtual memory size of each slot's machine $\in S_{x_i} \geq x_i$. The result will be a histogram. The membership function is generated using (2c) by connecting the second edge of each bar in the histogram to the next bar's using a straight line, and divide the result by the total number of slots in the domain, $|S|$ in order to keep $Y(x) < 1$. Three examples of the Image_size membership function are shown in Figure 3b.

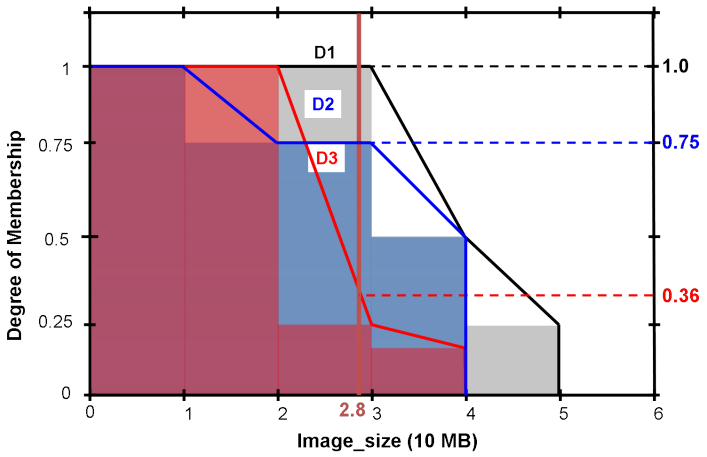
$$S_{D_x}(x_i) = \{s|\ s \in S_{D_x} \wedge vm(s) \geq x_i\} \tag{2a}$$

$$y_{D_x}(x) = |S_{D_x}(x_i)| \quad \forall x \in (x_{i-1}, x_i] \tag{2b}$$

$$Y_{D_x}(x) = \frac{1}{|S_{D_x}|}\left[\frac{(y_{D_x}(x_{i-1}) - y_{D_x}(x_i))(x - x_i)}{x_{i-1} - x_i} + y_{D_x}(x_i)\right] \quad \forall x \in (x_{i-1}, x_i] \tag{2c}$$



(a) ♯ CPU slots



(b) Image_size (MB)

Fig. 3: Fuzzy input membership functions

Fuzzification of Inputs The fuzzification process is to take the inputs, determine the degree to which they belong to each of the appropriate fuzzy sets

via input membership functions. Using the two inputs of Figure 2, 60 slots and 28 MB image size, the degree of membership in the three domains, D_{15}, D_{22}, and D_{20} is determined by the intersection as shown in Figure 3a.

Applying fuzzy operator Once the inputs have been fuzzified, we know the degree to which each part of the antecedent has been satisfied for each rule. A separate rule must be created for each domain, D_x in the match list of the matchmaking process for job j. The rule for N domains can be formulated as follows:

if $CPUSlotsOf(j)$ **is** $AllocatableInCPUSlots(D_i)$
and $MemoryImageOf(j)$ **is** $AllocatableInVirtualMemory(D_i)$
then j **is** $AllocatableIn(D_i)$

where $i = 1, 2, \ldots, N$ and D_i is the domain identifier. The **and** operator used in the rule above is implemented as a minimum function: V_1 **and** $V_2 = \min(V_1, V_2)$. Using the example of Figure 3, $membership_{D_{15}} = \min(1,0.63) = 0.63$.

Applying the Implication Method Before applying the implication method, we set each rule's weight. The rule weight of the fuzzy set associated with each domain D_x is calculated as:

$$RW(D_x) \;=\; \frac{1}{|r_{D_x}| \times (1 + \frac{q_{D_x}}{|S_{D_x}|})}$$

where $|r_{D_x}|$ is the number of hops in the shortest route from the local domain to D_x, q_{D_x} is the queue size of D_x, and $|S_{D_x}|$ is the total number of slots in D_x. Once proper weighting has been assigned to a rule, each of the resulted values from implementing the fuzzy operator in each rule is multiplied with its associated rule weight. The output membership function associated with each fuzzy set will be a single number representing the unique identifier of the associated domain (ID(D_x) i = 1,. . . , N). The outputs of all rules are aggregated into a single fuzzy set whose membership function assigns a weighting for every output value, so that the domain with the highest match will have the highest resulted weight.

Defuzzification The input to the defuzzification process is the aggregate output fuzzy set and the output is a single number. The fuzzy set is the collection of weights resulted from the previous step. In this approach, the maxima aggregate function is implemented. The active membership function, associated with N input fuzzy sets for N domains for matching with a job j, will be:

$\mathbf{OUT}$(j,N) = $\mathbf{MAX}[w(D_1), w(D_2), \ldots, w(D_N)]$

Where: $w(D_i)$ is the weight associated with the domain D_i, and $\mathbf{OUT}$(j,N) represents the value associated with the ID of the most suitable domain to allocate j. The final output is a List of IDs of the domains in descending according to the associated weights.

5 Performance Evaluation

We test the efficiency of four inter-grid scheduling techniques: UNICORE service orchestrator [8], Condor flocking [4], P2P Condor flocking [11], SLICK first-match [16], and SLICK best-match.

UNICORE service orchestrator is a meta-scheduler that manages job allocation between batch systems and flat-structured Grids, e.g. condor [19] and PBS [27], by installing a UNICORE *gateway* on each domain which is connected to the local broker [8]. Inter-grid communication and job submission in UNICORE is carried out in a central manner through the *service orchestrator*. Condor flocking is a client-to-domain interconnection. In case the local workers are busy, jobs can be submitted to external brokers which are listed in a manually configured list. No matching is performed before submission. Submission to external brokers is totally based on the order of the list. P2P condor flocking is a resource discovery technique which is built on the top of condor CentralManager, i.e. local broker, to enable interconnecting condor pools, i.e. domains, in a structured p2p overlay. For cross-domain submissions, Only two attributes are used to decide to which broker to submit two: average CPU utilisation of the workers in that domain, and the broker queue size.

We present the results of simulating a large number of domains using PeerSim peer-to-peer simulator [18]. We simulate a system of 50,000 nodes in 512 interconnected domains. The domains are connected through local brokers in a HyperCube logical topology, i.e. in case of N interconnected domains, each broker will have k neighbours in its routing table where $k = \frac{\ln N}{\ln 2}$. In case of 512 brokers, each broker will have 9 neighbours. We implement a HyperCube instead of a full Mesh since it uses a smaller neighbourhood degree k and at the same time achieves efficient data synchronisation, described in [28]. In case of UNICORE service orchestrator, a logical star topology is used since UNICORE implements central meta-scheduling. Worker machine specifications are of two machine configuration groups:

- group1: 2 CPU slots, 4GB Memory, Windows OS, No java support.
- group2: 4 CPU slots, 8GB Memory, Linux OS, Java support

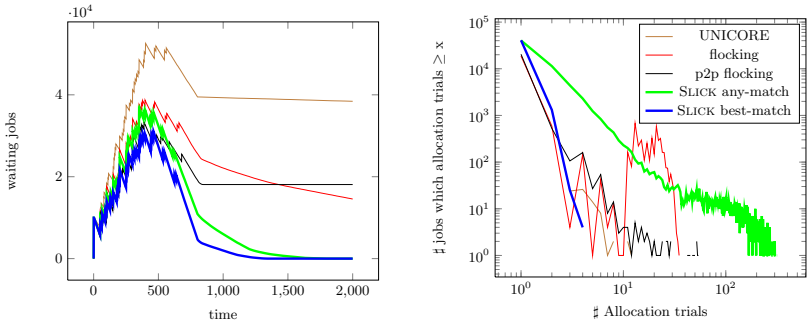
Workers are divided equally between the two groups, 25,000 each, but scattered among the domains. We create a load of total 80,000 synthetic jobs. Job resource requirements are randomly set to be matching either group1 or group2. The execution time of each job is uniformly distributed $t \in [100, 300]$ time units. Jobs are divided into 100 sequences. Each sequence is assigned to one broker. Using a uniformly distributed frequency $f \in [50, 100]$ time units, a uniformly distributed number of jobs $j \in [50, 100]$ is submitted periodically by each sequence. The process continues until all the 80,000 jobs are submitted. The total simulation time of the experiment is set to 2000 time units. The simulation is configured such that each of the following takes places in a one simulation time unit:

- The local scheduler processes one job from the local queue, i.e. either allocating the job in the local domain in case of local matching.

– The gateway scheduler processes one job from the gateway queue, in case of SLICK, i.e. allocating the job to the matching domain. The job transmission time is proportional to the route length from the local domain to the matching domain, e.g. if the route length is 3 hops then the job transmission time is 3 time units.
– Each SLICK broker synchronises GIS with one neighbour broker in the overlay. The GIS synchronisation algorithm is described in [16]. Due to the size reduction of the resource information, each GIS synchronisation takes only one time unit.

We use two benchmarks: *Job allocation throughput*, and *Load balancing*.

Job allocation throughput is measured by reading the total number of waiting jobs in the system/time, Figure 4a, and number of job allocation trials Figure 4b. In terms of decreasing the number of waiting jobs, it is clear that both SLICK any-match and best-match techniques manage to reach a steady state. The any-match manages to allocate all jobs within 1800 time units, while best-mach does in 1345 time units. This shows the positive influence of applying the new technique on the performance. With other systems, a bottleneck case happens. This can be described that: For the UNICORE case, this is due to the fact that there is only one central meta-scheduler to carryout interconnections, which for cross-domain submissions allocates only one job per time unit. In case of flocking and p2p flocking, the reason of the bottleneck is the difference in worker specifications. for flocking, the brokers in the flocking list may be already saturated and/or don't have matching workers. In p2p flocking, the best broker is chosen based on its queue size and average CPU utilisation of its domain. However, the domain with the least loaded queue and lowest CPU utilisation may not have matching workers in other terms, e.g. operating system. The breakdown in curves after $\approx$ 800 time units is because we configured all job sequences to complete their submissions by that time. This was done in order to validate the system performance when job allocation is carried out only inter-domain and not intra-domain. The job allocation trials benchmark, Figure 4b, tells how many jobs needed to be reallocated to another domain how many times in order to start running. The value of $y(x_i)$ is the number of jobs which needed to be reallocated x_i times. $y(0)$ is the number of jobs which have not been reallocated, i.e. were successfully allocated in their local domains. In case of SLICK best-match, the number of re-allocations doesn't exceed 4. +50% of jobs are reallocated once. This is a positive indication that almost all scheduling decisions are accurate. In case of SLICK any-match, the number of re-allocations exceeds 300 sometime. This is mainly because a matching domain doesn't necessarily has a number of idle slots matching a particular job in the time of this job's submission. All jobs are managed to find a suitable domains though. Other systems show different behaviours, but not much can be concluded since none of them manage to allocate all jobs. So, the displayed behaviour is not for a small portion of jobs.



(a) Total waiting jobs

(b) Number of allocation trials per job

Fig. 4: system throughput: Overall Job Allocation Performance

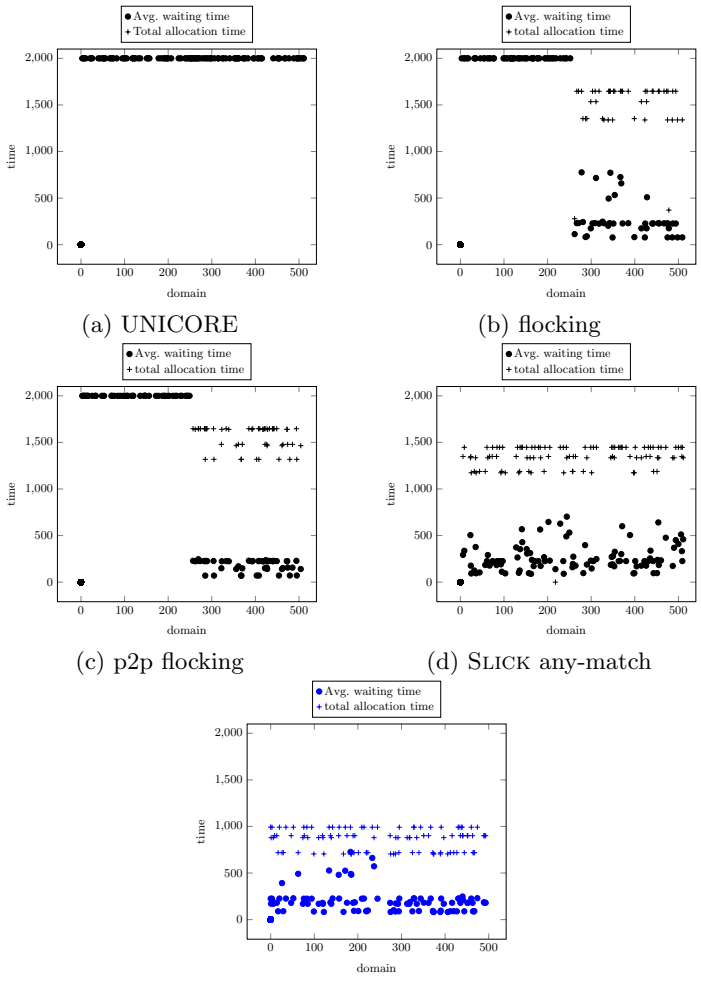
Load Balancing is measured by calculating for brokers, throughout the simulation: How long did it take to allocate all jobs owned by the domain of each broker, and what is the average waiting time. In Figure 5, it is clear that for SLICK any-match technique, the total allocation time never exceeded 1500 time units and the average waiting time is below 800. The output is even better in case of SLICK best-match, the total allocation time is below 1000, and the average waiting time is below 700. For UNICORE, none of the domains were able to finish the job allocation before the end of the simulation at 2000 time units. The same case occurs for more than half of the brokers in case of flocking and p2p flocking. The value of 2000 for both total allocation time and average waiting time indicates that this broker's jobs were not totally allocated.

6 Conclusions

This paper presented a coordinated scheduling technique for interconnected Grid domains. The key feature of the proposed technique is to allocate starving local jobs to the best-match domain, which is carried out by a fuzzy matchmaking. The proposed technique is tested through simulating its implementation on 512 interconnected Condor pools. It was proven that the proposed technique provides accurate cross-scheduling decisions compared to other techniques. Results also show that the proposed technique achieves load-balancing, high throughput, and is stable under high loads. We are planning to develop a failure control model for broker failures and churn. We are also planning to implement SLICK on a inter-grid prototype having brokers and workers as virtual machines on Amazon EC2.

References

1. I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International J. Supercomputer Applications*, vol. 15, no. 3,



(a) UNICORE

(b) flocking

(c) p2p flocking

(d) SLICK any-match

(e) SLICK best-match

Fig. 5: Load balancing: Average waiting time and total allocation time of jobs submitted to each of 512 domains

2001.
2. C. S. Yeo, R. Buyya, M. D. de Assuno, J. Yu, A. Sulistio, S. Venugopal, and M. Placek, *Utility Computing on Global Grids*. John Wiley and Sons, Inc., 2007, pp. 110–130.
3. R. Ranjan, "Coordinated resource provisioning in federated grids," Ph.D. dissertation, The University of Melbourne, Australia, July 2007.
4. X. Evers, J. F. C. M. de Jongh, R. Boontje, D. H. J. Epema, and R. van Dantzig, "Condor flocking: load sharing between pools of workstations," Department of Technical Mathematics and Informatics, Delft University of Technology, Delft, The Netherlands, Tech. Rep., 1993.
5. C. for High Throughput Computing, *HTCondor Version 7.9.3 Manual*, University of WisconsinMadison, Jan 16 2013.
6. E. L. et al., "Programming the Grid with gLite," CERN, Geneva, Tech. Rep., Mar 2006.
7. J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-g: A computation management agent for multi-institutional grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, July 2002.
8. B. S. et al., "Chemomentum - unicore 6 based infrastructure for complex applications in science and technology," in *Euro-Par Workshops*, 2007, pp. 82–93.
9. R. Buyya, D. Abramson, and J. Giddy, "Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid," in *InProcee. HPC ASIA 2000*, 2000, pp. 283–289.
10. J. Schopf, "Ten actions when superscheduling," In Global Grid Forum., 2001.
11. A. R. Butt, R. Zhang, and Y. C. Hu, "A self-organizing flock of condors," *Journal of Parallel and Distributed Computing*, vol. 66, no. 1, pp. 145 – 161, 2006.
12. J. B. Weissman and A. S. Grimshaw, "A federated model for scheduling in wide-area systems," in *Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing*, ser. HPDC '96. Washington, DC, USA: IEEE Computer Society, 1996, pp. 542–.
13. H. Shan, L. Oliker, and R. Biswas, "Job superscheduler architecture and performance in computational grid environments," in *In: Proc. of the 2003 ACM/IEEE Conference on Supercomputing (SC*, 2003, pp. 44–58.
14. C. Daval-Frerot, M. Lacroix, and H. Guyennet, "Federation of resource traders in objects-oriented distributed systems," in *Proceedings of the International Conference on Parallel Computing in Electrical Engineering*, ser. PARELEC '00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 84–.
15. K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. A. Huberman, "Tycoon: An implementation of a distributed, market-based resource allocation system," *Multiagent Grid Syst.*, vol. 1, no. 3, pp. 169–182, Aug. 2005.
16. A. Azab and H. Meling, "Slick: A coordinated job allocation technique for inter-grid architectures," in *7th European Modelling Symposium (EMS)*, Nov. 2013.
17. A. Vahdat and D. Becker, "Epidemic routing for partially connected ad hoc networks," Duke University, Tech. Rep., July 2000.
18. A. Montresor and M. Jelasity, "PeerSim: A scalable P2P simulator," in *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, Seattle, WA, Sep 2009, pp. 99–100.
19. M. Litzkow, M. Livny, and M. Mutka, "Condor - a hunter of idle workstations," in *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.
20. C. Aiftimiei, P. Andreetto, S. Bertocco, S. Dalla Fina, A. Dorigo, E. Frizziero, A. Gianelle, M. Marzolla, M. Mazzucato, M. Sgaravatto, S. Traldi, and L. Zan-

grando, "Design and implementation of the glite cream job management service," *Future Gener. Comput. Syst.*, vol. 26, no. 4, pp. 654–667, Apr 2010.
21. "NorduGrid: Nordic Testbed for Wide Area Computing and Data Handling – http://www.nordugrid.org/."
22. A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," *IN: MIDDLEWARE*, pp. 329–350, 2001.
23. I. Stoica, R. Morris, D. Liben-nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, pp. 17–32, 2003.
24. R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed resource management for high throughput computing," in *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, Chicago, IL, July 1998.
25. "2.5 Submitting a Job – http://research.cs.wisc.edu/htcondor/manual/v7.8/."
26. T. Takagi and M. Sugeno., "Fuzzy identification of systems and its applications to modeling and control," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 15, pp. 116–132, Jan. 1985.
27. B. Bode, D. M. Halstead, R. Kendall, Z. Lei, and D. Jackson, "The portable batch scheduler and the maui scheduler on linux clusters," in *Proceedings of the 4th annual Linux Showcase & Conference - Volume 4*, ser. ALS'00. Berkeley, CA, USA: USENIX Association, 2000, pp. 27–27.
28. A. Azab and H. Meling, "Decentralized service allocation in a broker overlay based grid," in *Proceedings of the 1st International Conference on Cloud Computing*, ser. CloudCom '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 200–211.