GA and PSO-based Resource Scheduling for Orchestrated, Distributed Computing

Yiran Gao[1], Chris Phillips[1], Liwen He[2]

Abstract: A new distributed computing architecture, Dynamic Virtual Private Network (DVPN), is introduced. The DVM (Dynamic VPN Manager) works as the Autonomous System (AS) administrator in the DVPN system to perform resource scheduling and liaise with the underlying connection management. The approach combines on-demand reservation of both the communications infrastructure and various higher-level processing facilities. This enables support of orchestrated computing where a complex job can be considered to be a VPN community. This job may be decomposed into tasks to be located at various distributed processing sites. Data can flow between them rather like a production line, in order to deliver the finished "product" to chosen end hosts. Two variants of a resource-scheduling algorithm are proposed for job scheduling in the DVPN system. Genetic Algorithm (GA) and Particle Swarm Optimisation (PSO) mechanisms are considered for use within the optimization process. Simulation results show that both approaches are feasible. The authors then compare the performance of GA against PSO in this dynamic VPN environment to compare their suitability.

Keywords: Grid Computing, GA, PSO, Resource Scheduling, Inter-Domain, VPN

1. Introduction

Layer-2 Virtual Private Networks (VPNs) can provide performance guaranteed communication for advanced IP applications such as Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), video-conferencing, and other mission-critical applications. However, typical VPNs are operated and managed via manual intervention, which prevents implementations of the VPNs from supporting new forms of application desiring dynamic community relationships and short-lived access to processing resources. Various researchers have focused on this limitation; for example, a number of dynamic VPN initiatives have emerged [1-8]. In this paper, the authors introduced a novel DVPN solution to extend the flexibility of inter-domain VPN operations with dynamic resources exploration and allocation in accordance with the customers' requirements. A new entity named the DVM (Dynamic VPN Manager) is introduced to provide enhancements to the MPLS signalling framework so that new functionality can be supported [9].

Two variants of a resource-scheduling algorithm are proposed to provide dynamic resource allocation in the DVPN system. Genetic Algorithm (GA) and Particle Swarm Optimisation (PSO) [10] mechanisms are employed as the optimisation tools. Their performance is examined through simulations.

The paper is organized as follows: Section 2 introduces the DVPN framework. Section 3 then describes the dynamic resource-scheduling algorithm. Next in Section 4 the performance of the proposed resource-scheduling algorithm is examined and a comparison is made between GA (Genetic Algorithm) and PSO (Particle swarm optimization) variants through simulations. Finally, conclusions are presented in Section 5.

2. The Dynamic VPN Architecture

Consider the scenario represented in Figure 1, where a job is launched by user A. This job consists of several sub tasks, T1, T2, T3, T4, T5 and the result will be transmitted to user D. According to the task dependence represented in Figure 1, T5 can start when T4 and T3 have both completed; T4 can start when T2 has completed, and T2 and T3 can start when T1 completed. Also, T3 is independent of T2 and T4. In this example, user A and user D are located in different domains (User A is in AS1 and User D in AS2). There are value-added processing resources available at B in AS1 and C in AS2.

Assume that this job is calculation intensive and extra processing resources are required to carry out the calculation tasks. Since some tasks are parallel such as T2 and T3, placing independent tasks on different processing resources may obtain better performance. In this example, a possible scheme is to process T2, T4 in resource B and T3 in resource C.

According to the above description, to process this job, a grid involving user A, user D, resource B and resource C is required. As mentioned before, a VPN can provide security and low cost links among the grid nodes. In order to use the network resources (i.e. bandwidth) efficiently, the VPN links (in the form of Label Switched Paths) are created only when the communications are necessary. After the data transitions are completed, the links are removed. At the same time, to better utilise the value-added resources such as processors, these resources can also be provided to the grid on-demand and dynamically. Meanwhile, the QoS requirements should be satisfied even when a VPN link crosses multiple domains (i.e. AS1 and AS2 in this example).

Traditional VPNs are static and based on human operations, which limits the scalability and the system dynamics. In the above scenario, to perform the grid computing job, given a traditional VPN, user A would need to send a request to the network manager to create a VPN and the VPN links would be configured by human operators. Once the VPN is created, all the VPN links and the network resources are allocated for the whole VPN lifetime. It is difficult to introduce new resources to speed up the grid computing calculation although there could still be available resources. Automatic releasing of the unused network resources is also impossible, because the network manager is a human operator, there is no such mechanism to manage VPN and network resource automatically. Meanwhile, in this example, the destination (user D) is in a different AS from user A. The VPN includes a link that must cross the border between AS1 and AS2, which requires complex processing for the traditional VPN technologies. It is also difficult for user A to utilise the resources at C because these entities within AS2 are invisible outside the domain.

Based on the above example, traditional technologies cannot efficiently provide dynamic and inter-domain services for the global computing. For this reason, the authors propose a novel

[1] Queen Mary, University of London, Mile End Road, London, United Kingdom
[2] BT Security Research Centre, Adastral Park, Ipswich, United Kingdom

inter-domain MPLS Dynamic VPN architecture to provide the inter-domain dynamic connectivity and resource management. Shown as Figure 1, the key contribution of the architecture is the inclusion of new equipment within each Autonomous System (AS), referred to as the DVM. Never the less, as far as possible, this architecture makes use of the existing forwarding and signalling methods or schemes under consideration in IETF RFC documents [8][11].

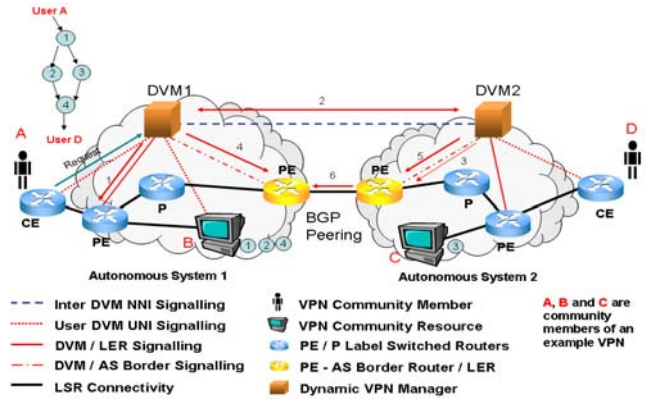


Figure 1: Dynamic VPN Architectural Overview

A DVM manages VPN communities under its jurisdiction. The user requests VPN services from the DVM. For example a user may request an MPLS connection between two sites where one of the sites hosts a processing resource. Data is to be sent from the first site to the second one for processing via an MPLS connection. When the processing is complete a second connection is temporarily established to return the results. The DVM will decide whether to provide this service and how to control the underlying network equipment to operate this MPLS VPN link. However, the DVM is not responsible for MPLS connection management and the formation of the Label Switched Paths (LSPs). This is the job of the separate connection management software existing within the operator's domain. The DVM merely identifies the member end-points and requests the connection management function to interconnect them via LSPs that have the desired QoS characteristics [9].

The end-point users may be human or CE-base software entities; they can request to create or join VPNs via the DVM User-Network Interface signalling. If the DVM accepts the user's request, it will coordinate the setting up LSPs among the user-end sites or some value-added resources. In such a system, the individual users can reserve / access computers, databases and experimental facilities on-demand, simply and transparently, without having to consider where those facilities are located. At the same time, the operators can offer the on-demand resource "farms" to satisfy user applications such as grid computing. This provides the operators with a source of revenue beyond the connection services alone, by leasing the value added resources as a part of DVPN services.

Figure 2 provides an example of how the orchestrated computing works in the DVPN system. The user submits the service requests to the DVM. The DVM makes the decision which resources will be employed to carry the tasks. DVMs will book the resource for the user's tasks. The VPN links will be setup among the users and resources dynamically for the data transactions. After that, the VPN link will be removed automatically to release the network resources. If the inter-domain services are necessary (i.e. data needs to be transferred to a resource located outside local AS), the DVM will negotiate with the other DVM. The inter-domain service will be provided to the customers automatically through the cooperation among the DVMs. However, the location of these remains transparent to the customers.

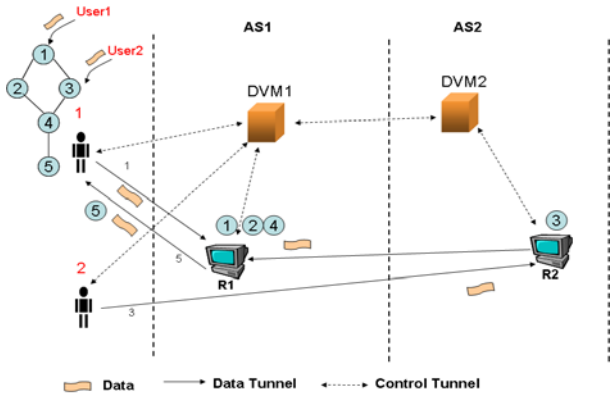


Figure 2: Orchestrated Computing in DVPN system

3. Resource Scheduling Algorithm

The DVPN architecture is designed to support services for applications that require access to on-demand and dynamic resources such as grid computing. A resource-scheduling algorithm is proposed to provide job scheduling in the DVPN environment.

A job will typically consist of a number of tasks. There might be the data dependences among these tasks. At the same time the jobs (VPN instances) are independent each other. Resource scheduling in such a heterogeneous system has been a well-known NP-hard problem [13]. A number of researchers have addressed the resource-scheduling problem for static cases of dependent / independent mixed jobs [14]][18] which are similar to the DVPN scenario. They are static in the sense that the characteristics of the jobs (i.e. arrival time, processing complexity) are known in advance. However, in the DVPN system, the job scenarios are dynamic and consist of dependent and independent tasks together where none of the existing static solutions are suitable. For this reason, a new dynamic resource-scheduling algorithm is proposed. The architecture of this scheme is represented in Figure 3.

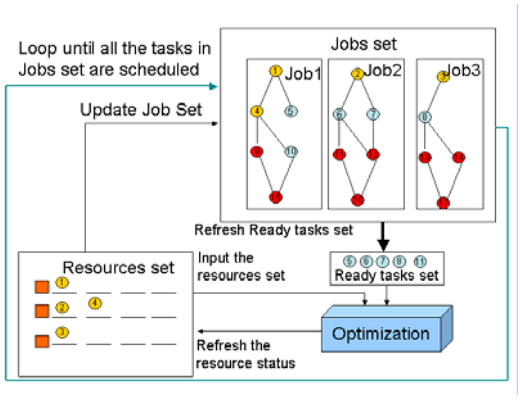


Figure 3: The Algorithm Architecture

The jobs are represented as Directed Acyclic Graphs (DAGs) and task dependency information of the jobs is stored in the

Jobs Set (JS). The Resource Set (RS) records the resource information including: availability, capacity and the currently allocated tasks.

The algorithm is designed to operate in dynamic scenarios where new tasks/jobs can arrive at any time. As there are typically dependencies among tasks comprising each job, the tasks are separated into different layers in accordance with their dependency constraints. The tasks in the succeeding lower layer are always compared with their preceding upper layer dependent tasks.

A task is "ready" for scheduling if all of its upper dependent tasks are already scheduled and their completion times are known. The scheduled tasks are the yellow nodes and the ready tasks are represented in blue. The red nodes indicate "un-ready" tasks that have some unresolved dependent uppers and so cannot be scheduled yet.

The ready tasks are input into the Ready Tasks Set (RTS). All the tasks in the RTS are independent because none of them have any un-scheduled dependent uppers, even though some of them may be associated with the same job. The ready tasks are scheduled using an optimization tool. In this paper, both a GA [19][20] and PSO [21] approach are considered as they are currently regarded as the robust and efficient stochastic searching mechanisms for various optimisation problems [22].

Whenever jobs arrive they are placed within the Jobs Set and arranged into layered tasks, bearing in mind their dependencies. From these jobs, tasks that have no dependent uppers may then be transferred to the RTS in preparation for a scheduling cycle. The scheduling processing cycle is initiated when there are tasks waiting in the RTS. While it executes, trying to find a suitable allocation of ready tasks to the available resources, no further tasks may enter the RTS. At the end of a scheduling cycle, the tasks in the RTS are assigned to the identified resources along with their start time. Also at this point the algorithm re-examines the Jobs Set, which may contain additional jobs that have arrived during the previous processing cycle as well as the residual tasks of the jobs that are currently being processed.

Tasks effectively enter the RTS in a step-wise fashion. Various mapping arrangements onto the available resources are attempted until the one with the smallest *job completion time* is found by the optimization engine. The task resource allocation information is updated, and then another scheduling cycle starts. The processing mechanism loops until all the tasks are scheduled, which means that no more tasks/jobs enter the network and the RTS is empty.

The advantage of this approach is that the optimization algorithm does not need complete information of all the jobs that will arrive; but only considers the tasks in the RTS, and possesses no foresight. The RTS updates in accordance with the current job information before each optimisation cycle starts. The algorithm can accommodate new jobs entering the system at any time. It is also possible that the resource availability can change whilst the algorithm is processing. The ability to accommodate these situations makes this algorithm feasible for use within the dynamic DVPN scenario and differentiates it from previous work.

4. Optimization with GA and PSO

In this paper, alternative GA and PSO optimization engines are considered for scheduling the ready tasks in the RTS at each resource-allocation cycle. Their performance is examined using simulations.

To illustrate the resource allocation mechanism and the role played by the optimisation algorithms, consider the trivial scenario depicted in Figure 4. This shows an example job dependency DAG a corresponding job scheduling arrangement in Figure 4 (a) and Figure 4 (b), respectively.

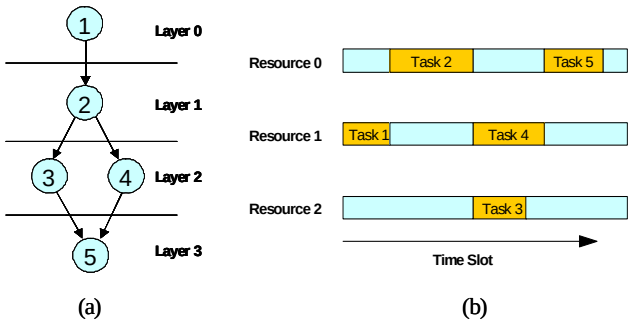


Figure 4: Example Job Scheduling Arrangement

In this example, a job is consists of five tasks and is represented as a DAG. There are 3 network resources. The tasks are separated into different layers and numbered in accordance with their specific dependencies as shown in Figure 4 (a). A task's number can never be smaller than the tasks in its upper layers. Also, a task cannot start until all its dependent uppers have finished execution. Considering the job scenario shown in Figure 4 (a), a possible scheduling arrangement might looks like Figure 4 (b).

1) GA implementation

GA [28][29] attempts to obtain an optimal solution through simulating the natural evolutionary process. The chromosome architecture employed in the proposed algorithm is illustrated in Figure 5. Each chromosome consists of a number of genes, where each gene represents a particular ready task awaiting scheduling. The value of the gene indicates the particular resource under consideration for allocating this task to. In the example of Figure 5, there are 5 genes in total, therefore representing 5 ready tasks. Assuming that there are 3 available resources (resources0, resources1, resources2), one allocation of the current ready tasks is shown in Table 1 in accordance with the chromosome of Figure 5.

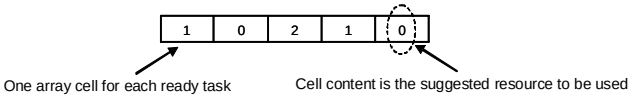


Figure 5: The Chromosome Architecture

Table 1: Relationship between the Tasks and the Resources Represented by Figure 5

Tasks to be scheduled	Available resources
Task 1	Resource 1
Task 2	Resource 0
Task 3	Resource2
Task 4	Resource1
Task 5	Resource 0

The chromosome only contains the mapping information of tasks to resources; however, the dependencies are considered when the fitness is calculated and tasks are assigned to resources.

The proposed GA-based dynamic algorithm was compared against a static algorithm. The static algorithm requires complete information concerning the whole scenario, specifically when each and every task will arrive and its burden. In the static case all the DAG tasks and the available resource information are read in before the optimization algorithm starts. The dynamic and static algorithms use GA as the optimization tool here. A similar chromosome arrangement is used in both of the two. The proposed algorithm outputs the scheduling arrangements for each set of ready tasks and terminates only when no more jobs enter the system. Conversely with the static scheme the final scheduling is output only once. This static algorithm provides a useful upper bound on the performance, as this complete knowledge of the job arrival sequence would not normally be available in a dynamic scenario.

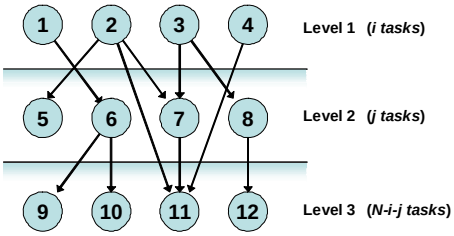


Figure 6: Job Scenario Example

In the simulation, the job scenario is generated as follows. Firstly, N tasks (for example N=100) are generated. Each task consumes a certain amount of the CPU resource. For example, if a task with task burden 4 is allocated to a resource with "capability" 2, the task will take two time units to complete. The burden of each task is uniformly randomly distributed (i.e. between 1 and 240). Then these tasks are separated into L levels (L=3 in Figure 6). The tasks 1 to i are located into level 1. i is a uniformly randomly-generated number between 1 and N-2. The tasks i to j are located into level 2, where j is a uniformly randomly-generated number between i+1 and N-1. The rest of the tasks are located in level 3. Thus there are i tasks in level 1 (Numbered from 1 to i), j-i tasks in level 2 (Numbered from i+1 to j), and N-i-j tasks in level 3 (Numbered from j+1 to N). The tasks in top level have no dependent uppers, and each task in the other levels (level2 and level3) has on average D (i.e. D=4, 10, 16, 22, 28 dependent uppers based upon the specific simulation scenario) that are Normally distributed (with mean=1, variance=1) dependent tasks in its preceding upper level. As a result, tasks are randomly arranged into several jobs that consist of a set of dependent tasks.

The number of generations for both algorithms is set to 10. Five job scenarios were examined. All of them contain 100 tasks while the average number of upper dependent tasks of each task is different in each case. For example, the green line demonstrates the simulation results under the scenario where each task has on average 4 dependent uppers; the black one is for the job scenario whose tasks have on average 28 dependent uppers.

From Figure 7 we can draw the conclusion that the dynamic algorithm's performance is similar to the static one in scenarios where the upper dependent task number is small, and the dynamic algorithm performs worse when the upper dependent task number of each task increases. As expected, the dynamic algorithm works well if the tasks are not very dependent, otherwise this algorithm will lead to a worse scheduling compared to the static case, where it has the luxury of a complete job forecast. The reason is that the dynamic scheme only considers the ready tasks, while the static one schedules according to the whole scenario; the dependent relations are considered along with knowledge of when all jobs will arrive. As a result, if these tasks are very dependent, the lack of the dependency information will lead to worse scheduling performance with the proposed algorithm.

Also from the Figure 7, we can find that both the static and dynamic GA-based algorithms achieve a shorter job completion time when the chromosome population number increases over a certain range. As previously pointed out, the task number in the RTS changes from time to time. Meanwhile, the GA's performance is related to the ratio of chromosome population versus the ready tasks, as implied by the Figure 7. Therefore, the GA algorithm was modified to adjust its chromosome population size according the current number of ready tasks in the RTS. The GA with an adaptive chromosome population size was compared against the one with fixed chromosome population size. The *chromosome population ratio* is introduced as the parameter representing the ratio of chromosomes to the number of ready tasks. For example, with the population ratio 0.5, the chromosome number should be 10 if there are currently 20 ready tasks waiting for scheduling. This adaptation ensures the complexity of the GA calculation is kept small when the presence of fewer ready tasks makes it appropriate to do so**.** Figure 8 presents the performance of the adaptive GA scheme.

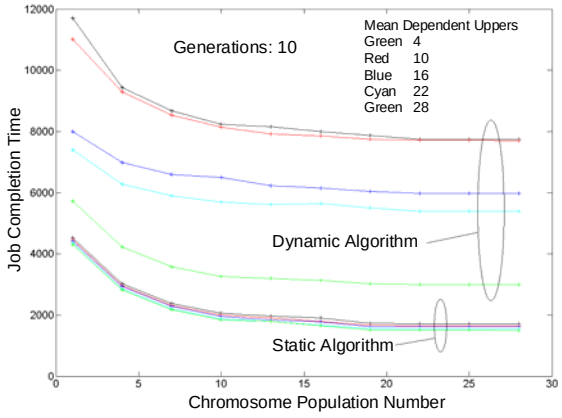


Figure 7: Job Completion Time Performance

In the simulation, the job scenarios each contain 100 tasks in total and 3 resources (with capacity 1, 2 and 3 respectively) are available. Each task has on average 10 dependent uppers. The horizontal axis is the consumed CPU computation time (i.e. how long the CPU spends calculating the scheduling arrangement) and the vertical axis is the chromosome population. Two simulations with the fixed chromosome population sizes (5 and 60) are carried out along with one simulation where the GA has the adaptive chromosome population size. As already defined, the population ratio is the ratio of chromosomes population size to number of tasks in the current assignment level. The population ratio was set to 1 for the adaptive chromosome population GA in this simulation, which means that the

chromosome population number always equals the current number of ready tasks. he simulation results show for the GA with a fixed chromosome population size (blue line in Figure 8), too small a fixed chromosome population number leads to poor scheduling, whilst too large a chromosome population number causes the CPU to spend too much CPU computation time calculating the assignment although its output scheduling is somewhat better than the GA with smaller chromosome population number.

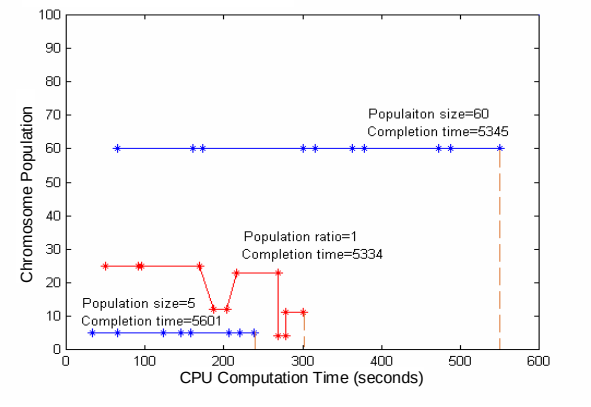


Figure 8: Adaptive Chromosome Population

On the other hand, the GA with adaptive-population size (red line in Figure 8) consumes less CPU computation time whilst outputting a similar scheduling result compared with the GA cases with large fixed chromosome population sizes. Its chromosome population size changes over time in order to maintain a relatively optimal chromosome population size according to the instantaneous ready tasks number. We can conclude that the GA with adaptive population size is more efficient than schemes with a fixed chromosome population size when considering the consumed CPU computation time and the resultant scheduling allocation.

Figure 9 examines the relationship between the chromosome population ratio and the scheduling allocation performance. Five scenarios with 20, 40, 60, 80 and 100 tasks were considered. Again there are 3 available resources (with capacity 1, 2 and 3). In each scenario, simulations with different chromosome population ratios were performed and 20 repetitions were carried out with the same chromosome population ratio (where a random number generator was used to create various job arrival permutations).

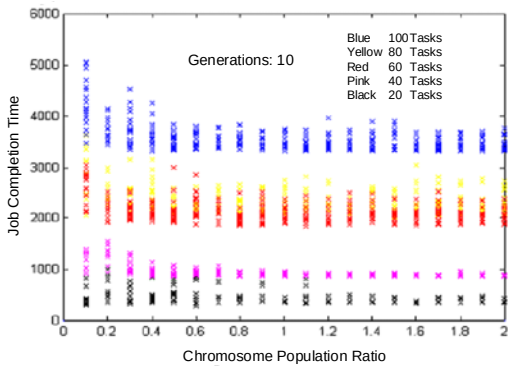


Figure 9: Chromosome Population Ratio Performance

Based on the results in Figure 9 we can see that increasing the chromosome population ratio over a certain range can improve the algorithm's scheduling performance and reduce the performance variation; however, further increases in the chromosome population ratio have a minimal impact.

As the chromosome population ratio increases, the time spent calculating the scheduling assignment also increases. The simulation results in Figure 10 show the relationship between the computation time and the job completion time for the proposed algorithm when the chromosome population ratio was varied from 0.1 to 2 (i.e. the same tests as used for Figure 9). As can be seen from the vertical axis, the range of job completion times achievable by different resource scheduling arrangements is limited in all cases. Also, especially when the RTS has more tasks for processing, increasing the chromosome population ratio can have a significant impact of the CPU computation time taken to arrive at a result for a given number of GA generations (i.e. from the increasing spread of values along the horizontal axis as the number of tasks mounts). This impact has little performance benefit as seen by the eventual plateau.

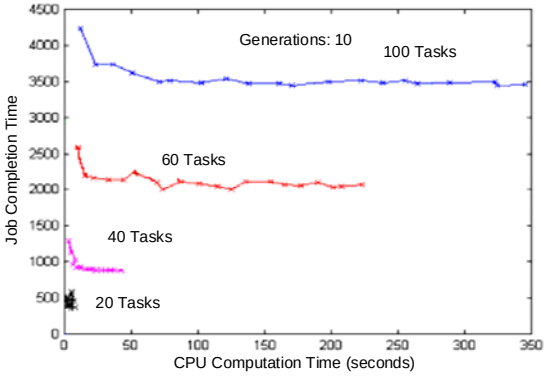


Figure 10: Job Completion Time versus Computing Time

2) PSO Implementation

Particle Swarm Optimization (PSO) is another evolutionary optimization technique. It simulates the process of a swarm of insects preying and works well in many global optimal problems [21][22].

PSO is employed as the alternative optimization tool. Similar with GA's chromosome, a PSO particle might be regard as one *n*-cell array where *n* equals the number of the tasks waiting for scheduling. Each cell maps a certain task and the cell value presented which resource this task is located. For example, the job scheduling arrangement shown in Figure 4 could be represented as the particle shown in Figure 11. The mapping relationship between the tasks and resources is demonstrated in Table 1.

{1, 0, 2, 1, 0}

Figure 11: Particle Example

The particle movement is achieved by swapping the number in each array cell which implies the resource-task pair. The velocity of the particle i in each dimension j, vij is determined by the number changed between the pij of last iteration to the next iteration. For example, if a particle's position is {0,1,2} and the

velocity is {1, 0, -1}, then the new position of this particle is {1,1,1}.

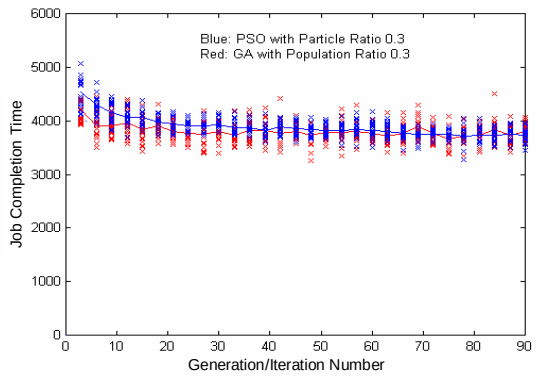
As with the GA-based approach, the number of particles can be adjusted in accordance with the current ready tasks number. The *particle ratio* is defined as the ratio of particles to the number of ready tasks.

3) Comparison between GA and PSO

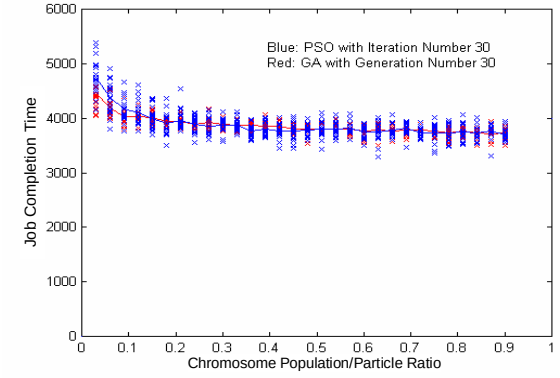
The authors compared the PSO performance against GA as the optimization engine with the same job scenario. 20 trials are carried out with the same chromosome population/particle ratio and generation/iteration number. The simulation results are shown in Figure 12.

In Figure 12 (a), the GA chromosome population and PSO particle ratios are maintained at 0.3 while in Figure 12 (b) the generation/iteration count is kept at 30 and various chromosome population/particle ratios are considered.

The PSO approach shows similar performance to the GA scheme. A shorter Job completion time is obtained by increasing the iteration count or the particle ratio over a certain range. Both GA and PSO converge close optimal solutions (shortest overall completion) if the chromosome population/particle ratio and generation/iteration number are large enough. However, GA can achieve a better solution when the chromosome population/particle ratio and generation/iteration number are small.



(a)



(b)

Figure 12:Performance Comparison between Static GA and Static PSO Resource Scheduling

Meanwhile, as the scheduling algorithm must operate in a dynamic environment, the speed of the optimisation engines must also be considered. Based on the simulation results given in Figure 13, with same chromosome population/particle ratio and generation/iteration number, it is seen that GA requires considerably more computation time than the equivalent PSO scheme.

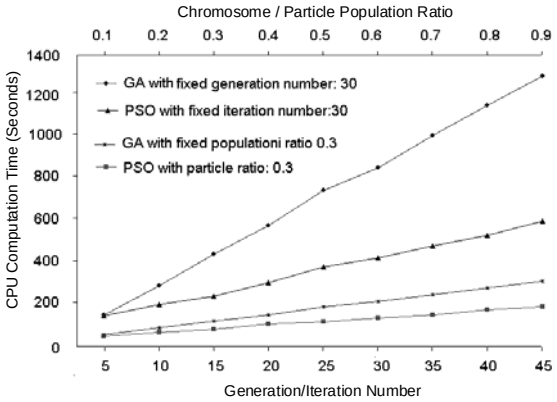


Figure 13: PSO and GA Computation Time Comparison

5. Inter-Domain Resource Allocation

As considered in Section 4, the DVPN architecture is designed to provide inter-domain services for orchestrated computing. Through negotiation among DVMs, available resources can be shared across the domain borders. Normally, using the inter-domain resources will incur extra cost, which is different from the local scenario. To make the dynamic resource scheduling algorithm proposed in this thesis also applicable to the inter-domains scenario, the fitness function needs to be extended to consider the inter-domain cost. In the following paragraphs, the inter-domain cost is introduced into the fitness function and the behaviours of the proposed algorithm under the multiple domains scenarios are studied through simulations.

The simulations are carried under a two-domain scenario shown in Figure 14. Only processing resources are considered. CPU 1, 2, 3 are located in AS1, CPU 4, 5, 6 are located in AS2. The capacity of each CPU resource is listed in Table 2.

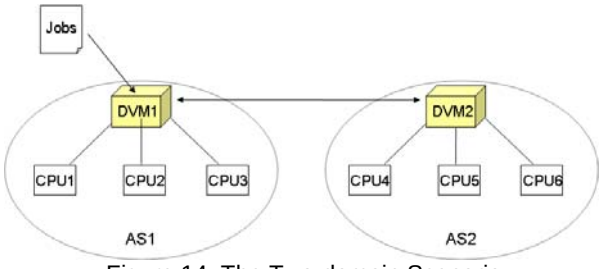


Figure 14: The Two-domain Scenario

Table 2: Resource Capacity

Resource Name	Capacity
CPU1	1
CPU2	2
CPU3	3
CPU4	1
CPU5	2
CPU6	3

During the simulation, DVM1 and DVM2 exchange the resource availability and keep resource information updated. For simplicity, the jobs are only submitted to DVM1 (the DVM of the AS1). Tasks allocated to CPU1, 2, 3 are performed locally; tasks allocated to CPU 4, 5, 6 are performed remotely, which incurs inter-domain cost. The inter-domain cost might take into account various factors, such as latency, inter-domain charging (fee paid for using resources), however, only the inter-domain charging fee is considered in these simulations and is represented as the cost in the simulation experiments. The fitness function is therefore extended as follows:

Fitness = [a×OverallCompletionTime)] + [b×OverallCost] …(1)

Where a is a time factor and b is a cost factor. Various values of a and b reflect how important the job Completion Time and Overall Cost are. For example, the bigger b is, the more important the inter-domain cost is regarded by the fitness function.

In the simulation, all the local resources cost 0 units and all the inter-domain resources cost 1 unit / burden. In this thesis, the cost is calculated as

Cost = (ResourceCapacity × Occupied Period) …(2)

Where the Occupied Period is the time that the resource spends carrying out this task.

The another Assumption is that all the jobs are submitted to the DVM in AS1, so if a task is carried out at any CPU resource in AS1, the cost will be 0; however, if a task is allocated to a CPU resource of AS2, an inter-domain cost is charged. For example, assuming the burden of a task is 1 the cost will be 1x1=1, if the burden is 10, then the cost for this task is 10x1=10 credit units.

In the simulation, the total task number is set to 100, the mean task burden is 240 (normal distribution, variance 100) and each task has 5 dependent uppers on average (normal distribution, variance=1). In each simulation trial, the generation number is set to 30 and the chromosome population ratio is set to 1.

In the first simulation experiment, the time factor “a” is always set to 1, the cost factor “b” is set to 0, which simulates the scenario where there is no inter-domain cost. The simulation results are shown in Figure 15.

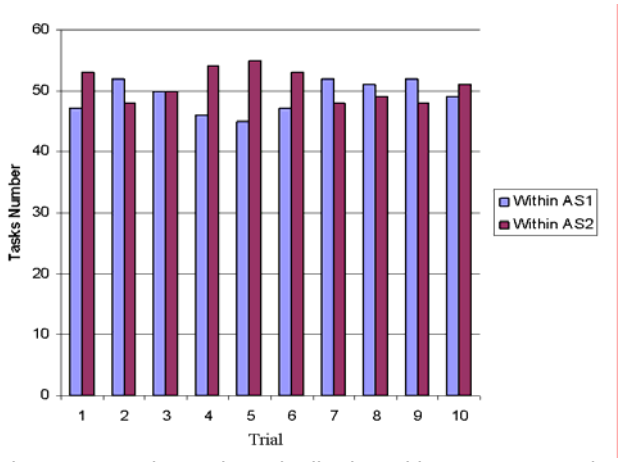


Figure 15: Task Number Distribution with no Inter-Domain Penalty(Inter-Domain Cost b=0)

10 independent trials are carried out. The total task number located within AS1 and AS2 are the same on average. The conclusion can be drawn that the algorithm treats the local and remote resources the same in this scenario where the inter-domain cost is 0, as expected.

In the second experiment, the inter-domain cost is considered. The cost factor b is set to 1. The simulation results are represented in Figure 16. As the inter-domain cost is considered when the fitness is calculated, the number of tasks located within AS1 (local AS) are more than the tasks located within AS2. This is because that the algorithm tries to avoid using the inter-domain resources which incur the extra cost. The average task number of 10 experiments is 65.300 for AS1 and 34.700 for AS2.

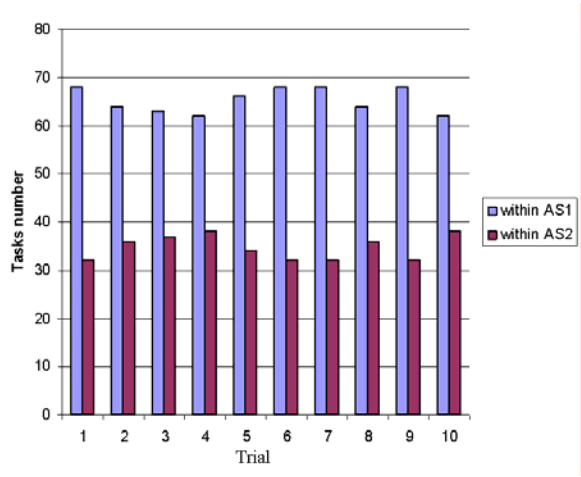


Figure 16: Task Number Distribution with Inter-Domain Penalty (Inter-Domain Cost b=1)

The relationship between the job completion time and the inter-domain cost factor b is also studied through simulations. Each simulation is repeated 20 times with same cost factor and the average job completion time is recorded. The results are shown in Figure 17, as the cost factor increases, the job completion time also increases. This is because a higher cost factor makes the cost a more significant factor for the fitness value. The algorithm is more likely to allocate tasks locally when the inter-domain cost is higher. Meanwhile the upper bound of the job completion time is same as the scenario where only local resources are available, because only the local resources will be employed if the cost factor is very large.

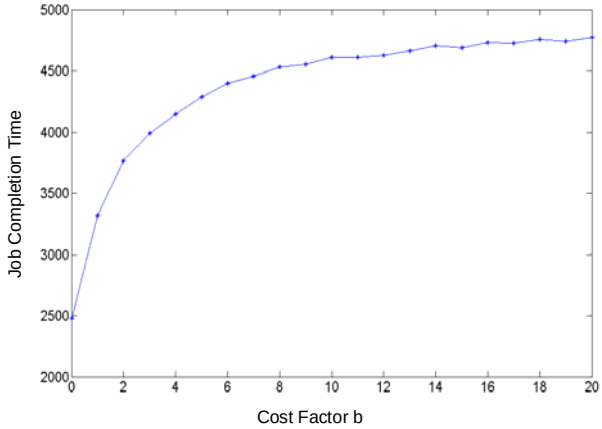


Figure 17: The Job Completion Time versus Cost Factor b

According to the above simulation results, the conclusion can be drawn that the dynamic resource scheduling algorithm is also appropriate for the inter-domain scenario. By introducing the inter-domain cost into the fitness function, a DVPN operator can regulate the extent to which inter-domain resources are employed for a given offered task load relative to the estimated job completion time.

6. Conclusions

A dynamic VPN architecture is introduced and two variations of a novel dynamic resource-scheduling algorithm are proposed. The algorithms are examined using simulations. The results show that both algorithms are feasible and should operate efficiently within a typical DVPN scenario. The GA and PSO optimisation engines are also compared. Both of them converge to similar overall job completion times when a large generation/iteration number and a sufficient chromosome population/particle ratio are used. However, with a limited generation/iteration number and a small chromosome population/particle ratio, the GA approach can typically obtain better results although the PSO scheme requires a much smaller computation time. Meanwhile, by introducing the inter-domain cost into the fitness function, a DVPN operator can regulate the extent to which inter-domain resources are employed for a given offered task load relative to the estimated job completion time.

References

1. Kindred, D.; Sterne, D. “Dynamic VPN communities: implementation and experience”, DARPA Information Survivability Conference & Exposition II, 2001. DISCEX '01. Proceedings Volume 1, 12-14 June 2001 Page(s):254 - 263 vol.1
2. Y. Jia et al, “Dynamic resource allocation in QoS-enabled/MPLS supported virtual private networks and its Linux based implementation”, IEEE CCECE 2002, Volume 3, 12-15 May 2002 Page(s):1448 - 1454 vol.3
3. Refer to: http://www.leetnet.org/ (URL accessed 20th June 2005)
4. Refer to: http://www.nrns.ca/DRDC.htm (URL accessed 20th June 2005)
5. P. Lago, R. Scandariato, “A TINA-based solution for dynamic VPN Provisioning on heterogeneous Networks”, Proc. IEEE Telecommunications Information Networking Architecture Conference (TINA'2000), Paris, France, 13-15 Sep. 2000.
6. Rebecca Isaacs, and Ian Leslie, “Support for Resource-Assured and Dynamic Virtual Private Networks”. IEEE JSAC, Vol. 19, No. 3, March 2001.
7. Fujita, N. et al, “Scalable overlay network deployment for dynamic collaborative groups”, Proc. 2005 Symposium on Applications and the Internet, Page(s):102 – 109, 2005.
8. L. Andersson, T. Madsen, “Provider Provisioned Virtual Private Network (VPN) Terminology”, IETF Request for Comments: 4026, March 2005.
9. Yiran Gao, Chris Phillips, Liwen He, “DVM Based Dynamic VPN Architecture for Group Working and Orchestrated Distributed Computing”, Third IEEE International Conference on Digital Information Management (ICDIM 2008), London, November 2008.
10. Lei Zhang1, et al, “A Task Scheduling Algorithm Based on PSO for Grid Computing”, International Journal of Computational Intelligence Research, ISSN 0973-1873 Vol.4, No.1 (2008), pp. 37–43.
11. E. Rosen, Y. Rekhter, “BGP/MPLS VPNs”, IETF Request for Comments: 2547, March 1999.
12. Muthucumaru Maheswaran et al, “Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems”, 8th Heterogeneous Computing Workshop, 1999.
13. Ajith Abraham et al, “Heuristics for Scheduling Jobs on Computational Grids”, The 8th,IEEE International Conference on Advanced Computing and Communications (ADCOM 2000), 2000.
14. Juan Antonio Gonzalez et al, “A Hyper-heuristic for scheduling independent jobs in Computational Grids”, AEOLUS Workshop on Scheduling, 2007.
15. R. Armstrong, D. Hensgen, and T. Kidd, “The relative performance of various mapping algorithm is independent of sizable variance in run-time predictions,” 7th Heterogeneous Computing Workshop (HCW’ 98), pp. 79-87, March 1998.
16. R. Freund et al, “SmartNet: a scheduling framework for heterogeneous computing,” The International Symposium on Parallel Architectures, Algorithms, and Networks, Beijing, China, June 1996.
17. Ibarra and C. Kim, “Heuristic algorithms for scheduling independent tasks on non identical processors.” Journal of the ACM, 24(2):280-289, April 1977.
18. Tianchi Ma and Rajkumar Buyya “Critical-Path and Priority based Algorithms for Scheduling Workflows with Parameter Sweep Tasks on Global Grids”, SBAC-PAD, pp 251-258, 2005.
19. Edwin S . H . Hou, Member, IEEE, Ninvan Ansari, Member, IEEE, and Hong Ren, “A Genetic Algorithm for Multiprocessor Scheduling”, IEEE Transactions On Parallel And Distributed Systems. Vol. 5, No. 2, February 1994.
20. Y. Gao, C. Phillips, “A GA Based Real-time Resource Scheduling Algorithm”, IEEE ICTTA08, Syria, April 2008.
21. Kennedy J. and Eberhart R. “Swarm Intelligence”, Morgan Kaufmann, 2001.
22. Lei Zhang, et al, “A Task Scheduling Algorithm Based on PSO for Grid Computing”, International Journal of Computational Intelligence Research, ISSN 0973-1873 Vol.4, No.1 (2008), pp. 37–43.