Chapter 5

Performance Measurement

5.1 Fundamental Questions

The design of a scheduling algorithm includes four fundamental aspects (see Section 2.1). Among these, being the most important is the performance of the algorithm. Trying to measure the performance of a Grid scheduling algorithm can be a great challenge. The dynamic Grid character makes difficult the development of any kind of standard benchmarks. The number of different factors that have to be considered if someone tries to create such a standard benchmark can be extremely large. Before measuring the performance of the **xDCP$_C$** algorithm we will try to answer some basic questions as they are formulated in [37]:

- **What problem parameters can affect the performance of the algorithm?**

  The task graph topology influences the most the performance of the algorithm. Moreover some other parameters may react on the performance. These can be the number of edges that start from a node (number of successors), the number of processors available and the CCR (Communication to Computation) value. All these factors need to be considered and thus it is imperative to use a various range of structure graphs with numerous different parameter configurations. This is the only way to do an accurate evaluation of such an algorithm. We will use the above mentioned factors for the **xDCP$_C$** evaluation.

- **What are the important performance measures?**

  The quality of the schedule is the total execution time of the application and states the performance of a scheduling algorithm. That is what we will measure for the evaluation of the **xDCP$_C$** algorithm. We will use the Shuffle (see Section 3.2) algorithm to compare the scheduling length that it creates with the scheduling length that creates the **xDCP$_C$** algorithm. The difference between the two algorithm scheduling lengths is the gained

Profit. We measure the **Profit** variation for a big number of configuration scenarios.

5.2 Factors and Performance Metrics

The metric we use for the evaluation of the $\mathbf{xDCP_C}$ algorithm is the average value of the **Profit**, obtained after a 20 times simulation run for the same configuration settings under the same task graph. We denote this metric **AGP** (Average Gained Profit). Our scope will be to investigate the **AGP** value variation under different parameter and graph settings. We regard the performance of the Shuffle algorithm as a basic performance under a certain graph topology and configuration setting, as it is described in [8].

Another performance metric is the **E** (Effectiveness). It is the ratio (percentage) between the final scheduling length of the $\mathbf{xDCP_C}$ algorithms and the scheduling length that produces the Shuffle algorithm. Again the AE (Average Effectiveness) value is being calculated, for the same simulation settings and input graph. It is the average value after 20 simulation runs. In terms it is:

$$\mathbf{AE} = \left(1 - \frac{\sum_{\mathbf{i}=1..20} \mathbf{xDCP_C lenght}}{\sum_{\mathbf{i}=1..20} \mathbf{Shufflelenght}}\right) \mathbf{x} 100\%$$

The subtasks weights are creating randomly using the given function:

$$Comp\_Size = rand(1, k) \; x \; base_1$$

We can vary the **k** value generating more or less computation demanding applications. The $\mathbf{base_1}$ value is a constant and defines the smaller computation size allowed in any subtask.

The data files weights are influenced by the given CCR value. For our experiment we use 2 different values of CCR (0.01 and 0.1). A DAG with CCR smaller than 1 is considered as coarse grain (low communicating application) and with CCR greater or equal than 1 as fine grain (heavy communicating application). Thus we use these 2 values to cover both aspects. For a given node, the size of the data file that it creates and sends to his successor(s) after its execution will be:

For CCR = 0.01 $Data\_Size(t) = Comp\_Size(t)/100$
For CCR = 0.1 $Data\_Size(t) = Comp\_Size(t)/10$

The size of every heterogeneous cluster $\mathbf{N(R_i)}$ is also generating randomly following the function:

$$N(R_i) = rand(1, l) \times base_2$$

Again we can vary the **l** value creating smaller or bigger clusters but we will keep constant the $base_2$ value defining this way the smaller allowed cluster for the experiment. For machines (CPU's) speed we rely on the model influenced by the Moore's law as it is discussed in Section 4.5. The links speed is also generated randomly. The difference between the slower and faster link is given by an increasing factor of 10 (e.g. slower 2 Mb/sec and faster 20 Mb/sec).

5.3 Input Task Graph Generation

A precise evaluation of the algorithm can not be done without using a wide range of input task graphs. Thus we used five different types of graphs. All the input graph weights are being valued using the factors mentioned in Section 5.2. An analytically description of each graph that is used follows:

(a) PTG (Parallel Task Graph)

In a PTG the number of subtasks on every layer is the same constant value. We generate random PTG's by defining the number of tasks on the first layer and the total number of layers. A paradigm of a PTG graph follows:

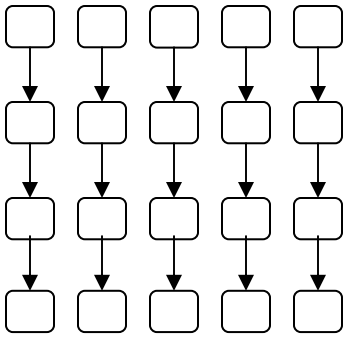


Figure 5.1: Parallel Task Graph

(b) OTG (Out-Tree Task Graph)

An OTG is created by attaching nodes starting from the entry (only one entry) we respect to the branch number (λ). The branch states the number of successors that a current node has. This value is constant for all the graph nodes. The number of subtasks on each layer is given by the term $\lambda^{(i-1)}$. For the experiment we generate random OTG graphs for various (λ) and layer values.

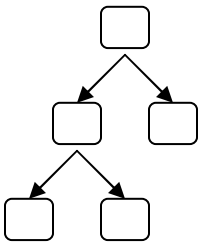


Figure 5.2: OTG (λ=2)

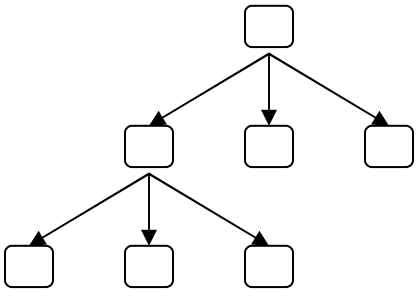


Figure 5.3: OTG (λ=3)

(c) ITG (In-Tree Task Graph)

The ITG graph is generated starting from the exit nodes, an opposite than for OTG way. Now the branch number (λ) does not state the successors but the predecessors of a current node. Thus the number of subtask on the first layer is given by the term $\lambda^{(\mu-i)}$ where **μ** is equal to the maximum number of layers and (i) is the number of the current layer. We generate random ITG's varying the (λ) and (μ) value.

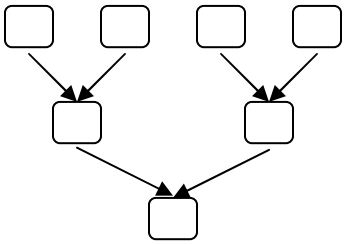


Figure 5.4: ITG (λ=2)

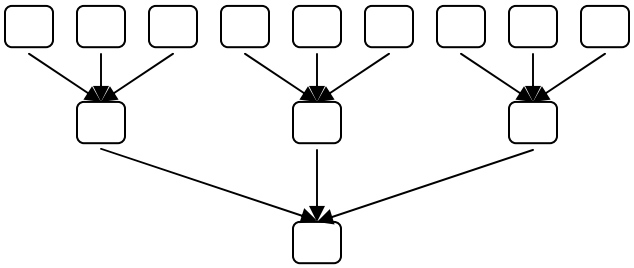


Figure 5.5: ITG (λ=3)

(d) DOTG (Densified Out-Tree Graph)

The creation of a DOTG starts by defining the number of subtasks on the first layer (e.g. N_1=2). The number of subtasks on the next layer can be calculated recursively starting from the subtasks on the firsts layer by the function $N_\iota = \delta(N_{(i-1)}-1) + \lambda$. The factor δ describes the allowed step value and the λ is the branch value, being again constant for all the nodes. We generate random DOTG's by defining the number of subtasks on the first layer and the number of application layers. We use for step (δ) the value of 1 and for branch (λ) the value of 2 for our experiment.

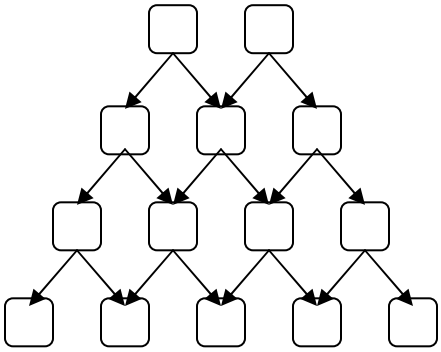


Figure 5.6: DOTG (λ=2, δ=1, N_1=2**)**

(e) DITG (Densified In-Tree Graph)

A DITG is created emerging predecessor tasks to create one successor. The (λ) value defines the number of predecessors that is needed to create one successor. The number of subtasks on each parameter sweep task (level) is defined recursively by the function $Ni = (N_{i-1}-\lambda)/\delta + 1$ starting from an initial first layer value (N_1). The values that we use is 2 for (λ), 1 for the step (δ) and various for N_1.

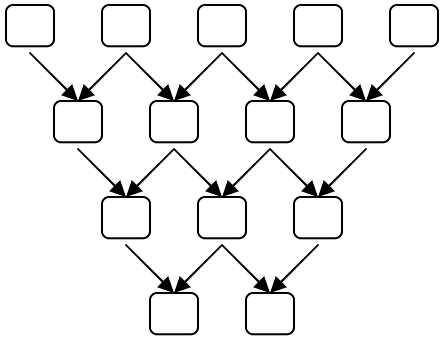


Figure 5.6: DITG (λ=2, δ=1, N_1=5)

Using all the above mentioned cases (a), (b), (c), (d), (e) we will cover a quite wide range of possible PST applications. Of course more graphs can be used as input for a more precisely evaluation of the algorithm. This can be scheduled as a future work.

5.4 Performance Evaluation

For the evaluation of the new proposed algorithm we will use the two performance metrics described in Section 5.2. Combining the factors we discussed in Section 5.2 and the input task graphs (see Section 5.3) we can build a simulation configuration table.

Graph	CCR	k	$base_1$	l	$base_2$	λ	δ	μ
PTG	**0.01-0.1**	**5**	**100**	**3**	**10**	**1**	**-**	**1...10**
OTG	**0.01-0.1**	**5**	**100**	**3**	**10**	**2-3-4**	**-**	**2...9**
ITG	**0.01-0.1**	**5**	**100**	**3**	**10**	**2-3-4**	**-**	**2...9**
DITG	**0.01-0.1**	**5**	**100**	**3**	**10**	**2**	**1**	**2...10**
DOTG	**0.01-0.1**	**5**	**100**	**3**	**10**	**2**	**1**	**2...10**

Table: Simulation Configuration Settings

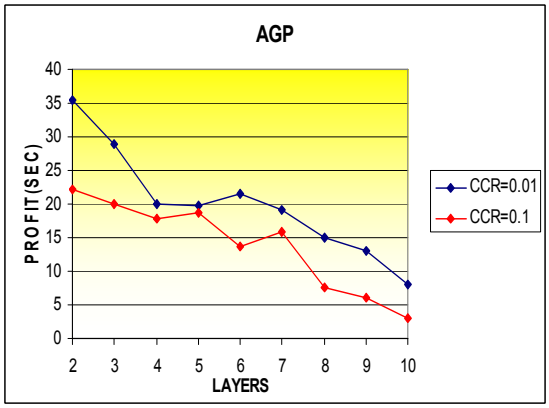
As the reader can easily observe there are 8 different factors and 5 input graphs considered for the simulation. The variation of these factors can create a huge number of possible simulation runs. In our evaluation we keep constant the **k, base1, base2** and **l**

values (see Section 5.2) and we vary the **CCR, λ** and **μ** values (see Section 5.3). We will investigate the impact of the variation of file, layer size and graph structure on the performance of the algorithm. We validate these factors as the most important because they have the biggest influence on the performance of the algorithm. The total number of simulation runs we performed overcomes the 2000 and the maximum number of tasks allowed is less or equal to 550.

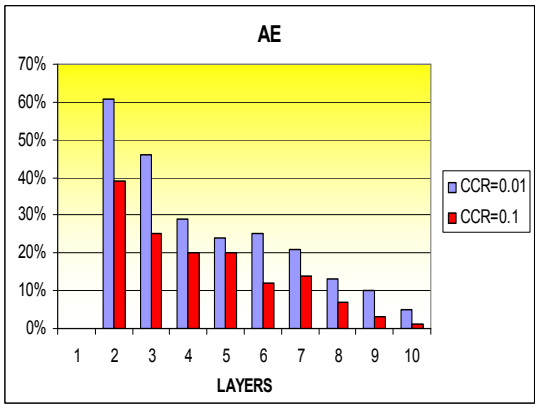
For convenience we mention that the variation of the AGP value is sometimes different from the variation of AE for the same case of graph. By increasing the layer size increases the total subtask number. The consequence is a higher in time units (sec) makespan. The rate of the **Effectiveness** can be lower (now we compute the rate on bigger weight sizes) but the **Profit** value may still stay within or near a high rate. For example if the scheduling length produced by the **Shuffle** algorithm is 500 sec and this created by the **xDCP$_C$** is 450 sec the profit would be 500-450=50 sec. The same profit we might obtain if the first algorithm produced 300 sec length and the second 250 sec length. Despite that the effectiveness would be 1-(450/500) = 10% for the first case and 1-(200/250) = 20% for the second case. Thus lower effectiveness might sometimes create a higher or similar AGP value than a higher one.

5.4.1 Parallel Task Graph

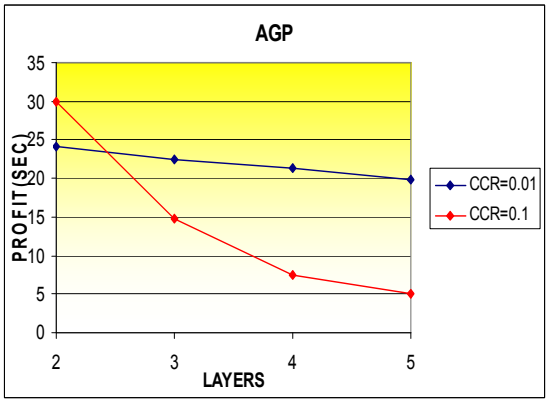
Two different graph structures are generated for this particular type of graph. One with initial subtasks number **N_1=50** and one with **N_1=100**. The maximum number of subtasks that is allowed is 500. This is proportional to a total number of 10 layers for the first case (Plot 1&2) and 5 layers for the second (Plot 3&4). In both cases we vary the CCR value.



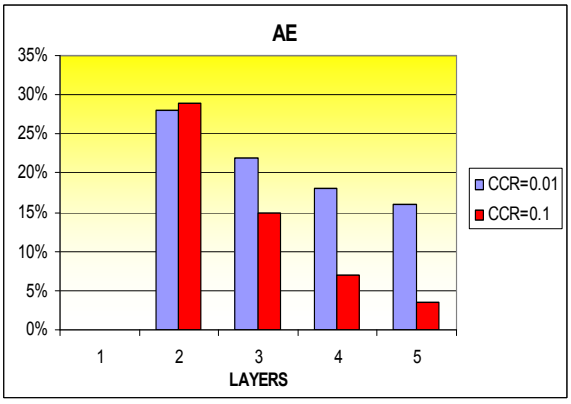
Plot 1: PTG (N_1=50)



Plot 2: PTG (N_1=50)



Plot 3: PTG (N_1=100)

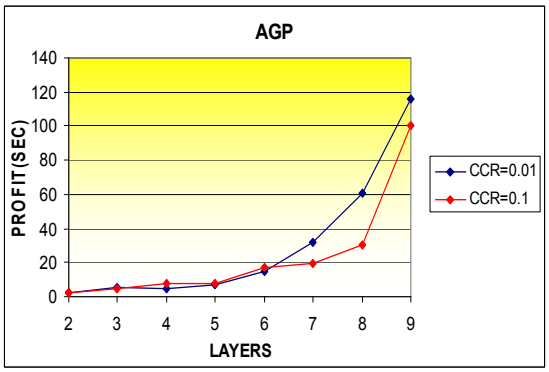


Plot 4: PTG (N_1=100)

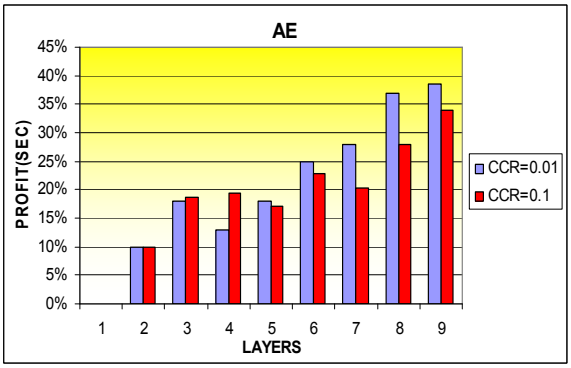
The result plots for PTG's show that the performance of the algorithm drops when the layer size increased for both cases (N_1=50, N_1=100). The same observation can be made when varying the CCR value from 0.01 to 0.1. This can be easily observed from the decreasing slope in all the above AGP plots. In this type of graphs there is only one successor for every subtask and so only one dependency. The CP of the graph does not change in every scheduling step. The **xDCP$_C$** algorithm tracks the changes on the CP, something that will never happen in the case of a PTG. Thus the effectiveness of the algorithm drops when the layer size is increasing. Consider the scenario of a PST graph that consists of 500 subtasks. This is a total of 10 layers for the case of N_1=50 and 5 layers for the case of N_1=100. The result AE for the first case (Plot 2) is 5% and for the second case (Plot 4) is 16% (for CCR=0.01). There is a 9% difference in the performance for the same subtask size application but for different graph structure. This difference is less easy to observe in the case of CCR = 0.1 graphs because of the negative influence that has the insertion of higher communication cost (consider that we keep the same link speed values).

5.4.2 Out-tree Task Graph

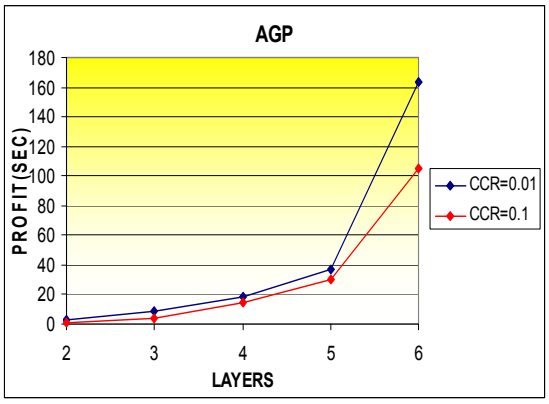
For the case of OTG's we use 3 different **λ** values (2, 3 and 4) to generate the task graphs needed for the experiment. The maximum number of subtasks allowed is 512 (scenario λ=2, μ=9).



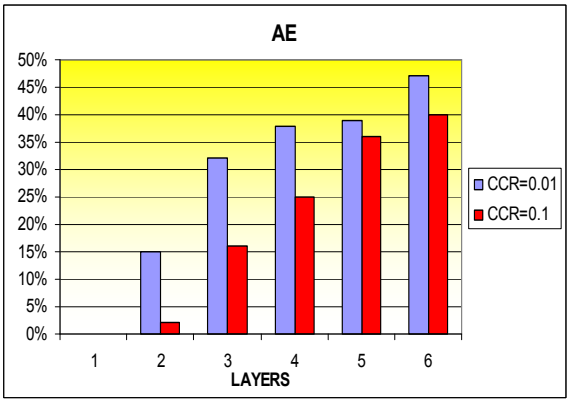
Plot 5: OTG (λ=2)



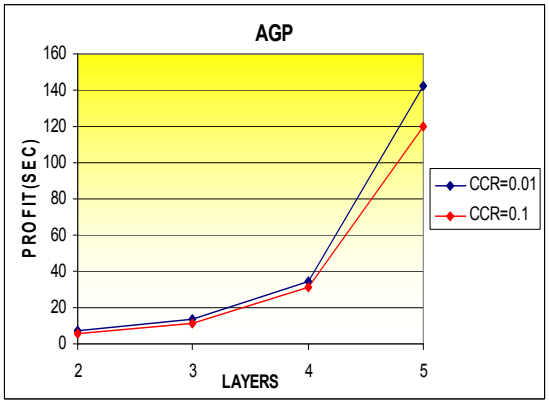
Plot 6: OTG (λ=2)



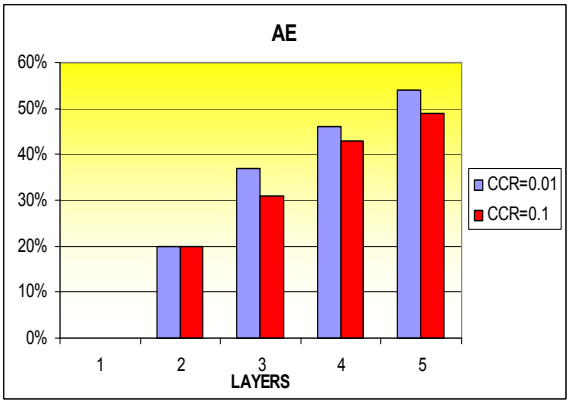
Plot 7: OTG (λ=3)



Plot 8: OTG (λ=3)



Plot 9: OTG (λ=4)



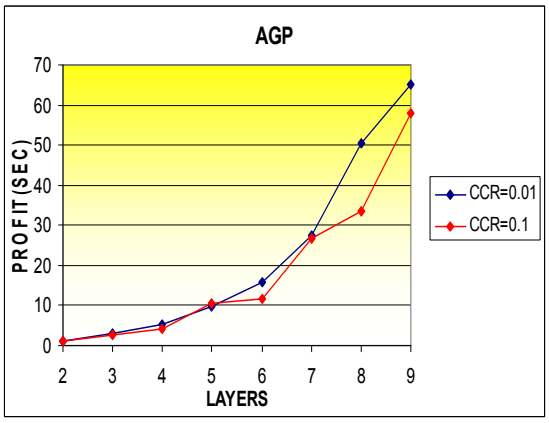
Plot 10: OTG (λ=4)

For OTG's (see Section 5.4.2) the performance of the algorithm increases when increasing the layer size. This happens because the shuffle algorithm leaves more gaps when the number of subtasks and layer increases. In the case of an OTG the CP may change after a scheduling step. The algorithm perform better than the PTG's (see Section 5.4.1) case because it is now able to track these changes and to take correct decisions when assigning tasks to processors. The optimization space becomes bigger when the layer size increases. The sizes of graphs that are created for different λ values are incommensurable thus we do not try to make a comparison in the same way we did for the PTG case. It is clear from the result plots (AGP and AE) that the effectiveness of the algorithm raises when the λ value (number of successors per predecessor) increases. The $xDCP_C$ tries to minimize the ALFT value of a ready to be schedule subtask. In the calculation of this attribute the algorithm considers all the successors of the ready subtask. Bigger the successors number greater the optimization space available.

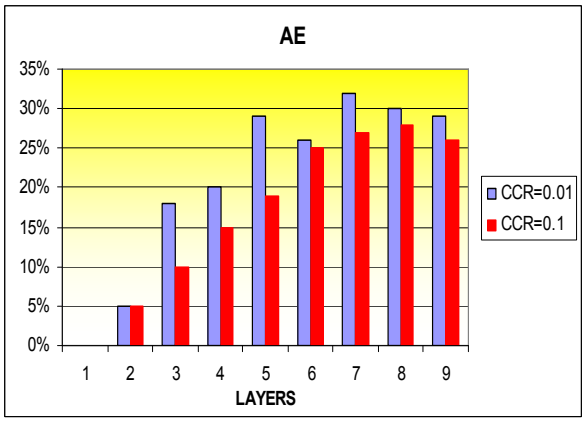
5.4.3 In-tree Task Graph

ITG's are being generated using 3 different λ values (2, 3, and 4). This type of graph is a reverse version of an OTG. Both ITG and OTG have the same total number of

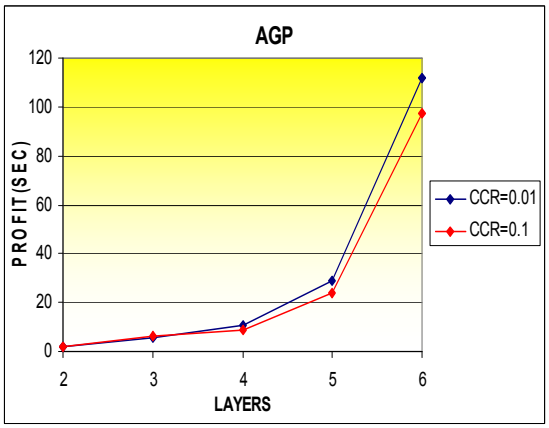
subtasks for the same λ and μ values. Again the maximum number of subtasks allowed is 512 (scenario λ=2, μ=9).



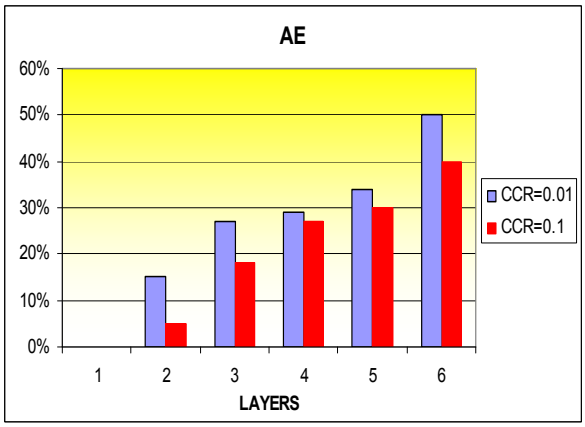
Plot 11: ITG (λ=2)



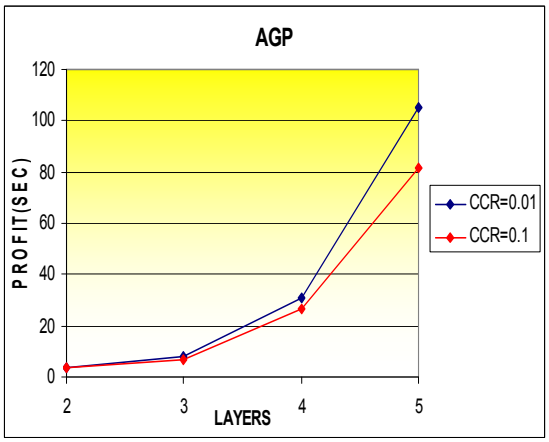
Plot 12: ITG (λ=2)



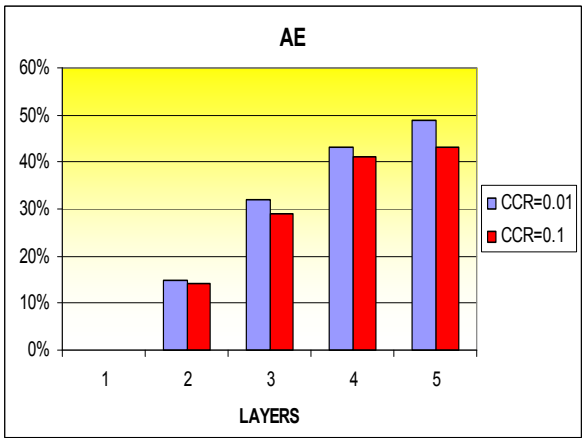
Plot 13: ITG (λ=3)



Plot 14: ITG (λ=3)



Plot 15: ITG (λ=4)



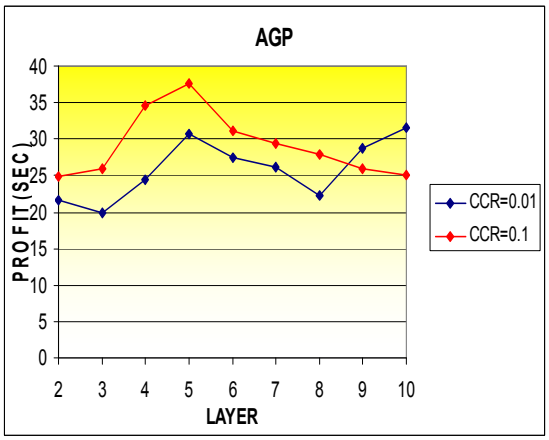
Plot 16: ITG (λ=4)

From the result plots one can observe that the algorithm performs worst than in an OTG (see Section 5.4.2) but better than a PTG (see Section 5.4.1). For scenario (λ=2,

μ=9, CCR=0.01) and an ITG graph (Plot 12), the effectiveness of the algorithm is 28%. For the same scenario but for an OTG graph (Plot 6) the effectiveness is 38%. There is a total 10% difference on the effectiveness for the same configuration settings and number of subtasks (total number 512 subtasks for both cases). If we consider an application with higher communication cost (e.g. scenario with λ=3, μ=6, CCR=0.1) then the effectiveness for ITG (Plot 14) is 40% and for OTG (Plot 8) is 48%. Again there is a total 8% difference in the performance of the algorithm. In ITG's there is only one successor subtask for various constant number of predecessor subtasks. The λ value defines the number of predecessors. Thus the ALFT value of λ number predecessors subtasks depends only on one successor subtask. The effectiveness of the algorithm increases with the increment of the λ value. The algorithm is now able to consider more subtasks in the calculation of the ALFT thus the probability for optimization is higher.

5.4.4 Densified Out-tree Task Graph

For the case of DOTG we use 2 N_1 values (50 and 100) to generate the graphs. The maximum number of subtasks allowed is 545.

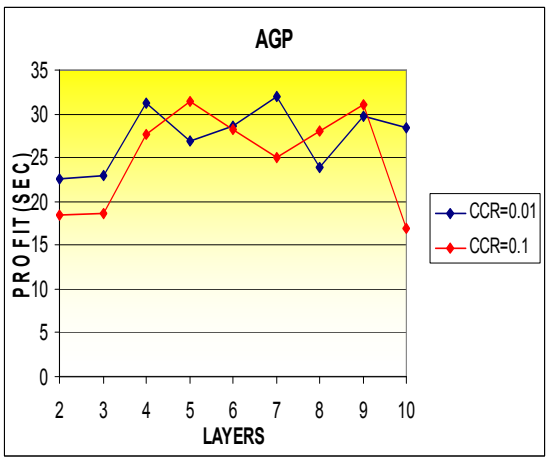


The performance of DOTG's is comparable to OTG's for a limited number of layers. For example for a DOTG with N_l = 50, μ = 5 and CCR=0.01 (Plot 18) the total number of subtasks that compose the graph will be 250 and the effectiveness is 32%. This number is almost the same for an OTG (Plot 6) with λ=2 and μ=8 with total number of subtasks equal to 255 and effectiveness 37%. The small difference on their effectiveness is due to the number of dependencies per subtask in a DOTG layer. This number is not constant. The outline subtasks have dependency 1 and the inline 2 (see Section 5.3 (d)). This discrete characteristic influences negatively the effectiveness of the algorithm. For a high number of layers this is easier to realize.

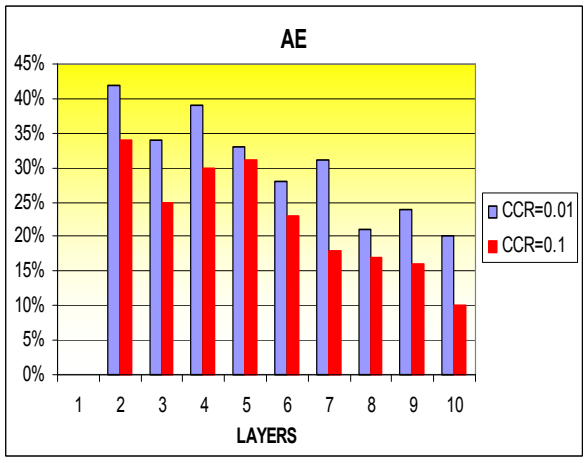
For a scenario (λ=2, μ=9, CCR=0.01) and for OTG graph (Plot 6) the effectiveness is about 38% and the number of subtasks is 512. Almost the same number of subtasks includes a DOTG with (N_l=50, μ=9, CCR=0.01). There the effectiveness is just 20%. It is obvious that the structure of two different but with equivalent subtask number graphs produces dissimilar results. The increment of the layer value has a negative influence on the effectiveness of the **xDCP$_C$** in the case of a DOTG. This is verified by the AE result plot values.

5.4.5 Densified In-tree Task Graph

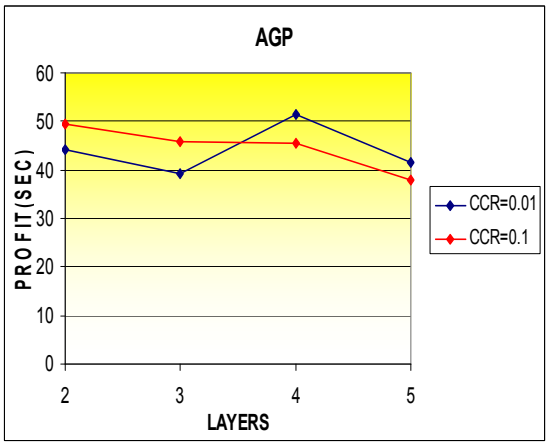
A DITG is constructed the other way around than a DOTG. We use the same values with the DOTG case, N_l=50 and N_l=100 to create the random DITG's for the two CCR values (0.01 and 0.1).



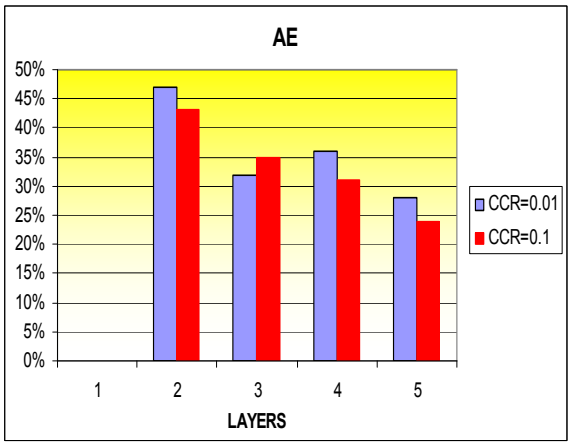
Plot 21: DITG (N_l=50)



Plot 22: DITG (N_l=50)



Plot 23: DITG (N_1=100)



Plot 24: DITG (N_1=100)

The effectiveness of the algorithm drops when increasing the layer size for both cases (N_1=50 and N_1=100). In the case DITG the algorithm performs with less effectiveness than for the case of a DOTG (see Section 5.4.4). This is something similar with the comparison of an OTG and ITG case. The result plots show to keep an analogy on the effectiveness with an OTG and an ITG respectively, at least for a high subtask number application (e.g. for μ=10, N_1=50 and μ=5, N_1=100).

Chapter 6

Conclusions and future work

This thesis describes the **xDCP$_C$**, a distributed algorithm for scheduling PST's in a Grid computing environment. Our goal was the insertion of communication cost within the structure of the **xDCP** algorithm. Besides that we did some more modifications to solve limitation problems we faced on the algorithm analysis procedure (see Section 3.4).

The result plots for all the above cases (PTG, OTG, ITG, DOTG, and DITG) prove that the graph structure influences the most the effectiveness of the algorithm. For applications with the same subtask number we find dissimilar effectiveness results (see section 5.4.3). Thus the answer we gave to the question in section 5.1 is proved to be correct. Moreover we observe that the algorithm performs better in the Out-tree Graphs than in the In-tree Graphs. The same observation can be made for the case of a DOTG and a DITG. The different kinds of dependencies that emerge in these particular types of graphs (see sections 5.4.4 and 5.4.5) have an individual influence on the performance of the algorithm.

Due to the lack of limited simulation runs (20 runs per layer value for every case), there is sometimes observed a greater variation on the AE and AGP value for some particular layer values. A higher number of simulation runs would likely create a smoother result. Of course this does not prevent us to make realistic conclusions about the performance of the algorithm. We reckon that 100 simulation runs per layer value would be enough to create a higher efficiency result. Such a number can reduce the probability to obtain similar, or near ranged random values (we used rand( ) to weight all the factors involved in the simulation). The various numbers of input graphs and factors that we used for the evaluation of the algorithm and the lack of time prevented us to use such a high number of simulation runs. A bigger number of input graphs with a higher number of simulation runs are sorely needed for a more precise evaluation of the algorithm. This can be scheduled as a future work.

There is one last question that we have to answer. This is:

"What was the impact of adding communication time cost into the xDCP algorithm structure?"

In order to answer this question we experimented with two CCR values (0.01 and 0.1) because we found wise to cover both a low and a heavy communicating application aspect. For the case of CCR = 0.01 (low communicating application) the results obtained keep an analogy with the results in [8], where the communication cost is not considered. Of course the link speed may have a very important role in the final schedule length. A heavy communicating application that uses high speed links can be equalized with a low communicating application that uses slow speed links. In our evaluation we kept the link speed within the same range for both cases (again we weighted the links randomly). Comparing the results we obtained for all the input graphs and both CCR values we find that high CCR creates lower effectiveness than a lower CCR value. That is what we where expected.

The impact of the network infrastructure is another factor that influences the performance of the algorithm and can raise or lower the communication cost of any PST application. In our implementation we assume that the links the user uses to communicate with the resource arrays (heterogeneous clusters) do not include routers. The current SimGrid API does not give us the flexibility to define more sufficient network cohesion. The new SimGrid API that is currently under development will include this flexibility and a higher efficiency network modeling can be a future work. It is necessary for a better evaluation to compare the **xDCP$_C$** with another Grid scheduling algorithm. Thus for future study we will try to modify the **pM-S** [8] algorithm. This is a dynamic algorithm that can be applied to a PST workflow but does not consider communication cost between the PST layers.

Bibliography

[1] Maurice Yarrow, Karen M. McCann, Rupak Biswas, Rob F. Van der Wijngaart "An Advanced User Interface Approach for Complex Parameter Study Process Specification on the Information Power Grid" Computer Sciences Corporation, Mail Stop T27A-1 NASA Ames Research Centre, Moffett Field, CA 94035 2003

[2] Abramson D, Giddy J, Kotler L. "High performance parametric modeling with Nimrod/G: Killer application for the global Grid?" Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2000). Cancun, Mexico, 1–5 May 2000 IEEE Computer Society Press: Los Alamitos 2000.

[3] SETI@HOME http://setiathome.berkeley.edu/

[4] J. Basney, R. Raman, and M. Livny. "High Throughput Monte Carlo" In Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing, March 1999

[5] T. Casavant, J.Kuhl, "Taxonomy of Scheduling in General-purpose Distributed Computing Systems", IEEE Trans. On Software Engineering Vol. 14, No.2, pp. 141- 154, 1998.

[6] Henri Casanova "Simgrid: a Toolkit for the Simulation of Application Scheduling" Computer Science and Engineering Department University of California, San Diego 2001

[7] J. D. Ullman, "NP-Complete Scheduling Problems", Journal of Computer and System Sciences 10, 384-393 (1975).

[8] Tianchi Ma and Rajkumar Buyya "Critical-Path and Priority based Algorithms for Scheduling Workflows with Parameter Sweep Tasks on Global Grids" Computing and Distributed Systems (GRIDS) Laboratory Department of Computer Science and Software Engineering the University of Melbourne, Australia 2005

[9] Wikipedia http://en.wikipedia.org/wiki/Directed_acyclic_graph

[10] Ishfaq Ahmad, Yu-Kwong Kwok Min-You "Performance Comparison of Algorithms for Static Scheduling of DAGs to Multiprocessors" Wu Department of Computer Science the Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong Sep. 1995.

[11] Yu-Kwong Kwok "High-Performance Algorithms for Compile-Time Scheduling of Parallel Processors", A Thesis Presented to the Hong Kong University of Science and Technology in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Computer Science Hong Kong May 1997

[12] Cooper, K. Dasgupta, A. Kennedy, K. Koelbel, C. Mandal, A. Marin, G. Mazina, M. Mellor-Crummey, J. Berman, F. Casanova, H. Chien, A. Dail, H. Liu, X. Olugbile, A. Sievert, O. Xia, H. Johnsson, L. Liu, B. Patel, M. Reed, D. Deng, W.Mendes, C.Shi, Z.YarKhan, A.Dongarra, J.Dept. of Comput. Sci., Rice Univ., Houston, TX, USA; "New grid scheduling and rescheduling methods in the GrADS project" International Journal of Parallel Programming 2005

[13] Fangpeng Dong and Selim G. Akl "Scheduling Algorithms for Grid Computing: State of the Art and Open Problems" Technical Report No. 2006-504 School of Computing, Queen's University Kingston, Ontario January 2006

[14] Tracy D. Braun, Howard Jay Siegel, Noah Beck Ladislau, L. Boloni, Muthucumaru Maheswaran and Albert I. Reuther "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems" Journal of Parallel and Distributed Computing 2001

[15] Yu-Kwong Kwok and Ishfaq Ahmad "Benchmarking and Comparison of the Task Graph Scheduling Algorithms" Department of Electrical and Electronic Engineering The University of Hong Kong, Pokfulam Road, Hong Kong Department Of Computer Science the Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong 1999

[16] Yu-Kwong Kwok and Ishfaq Ahmad "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors" the Hong Kong University of Science and Technology1999

[17] Howard Jay Siegel, Shoukat Ali "Techniques for mapping tasks to machines in Heterogeneous computing systems" Journal of Systems Architecture 46 627±639 School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-1285, USA 2000

[18] Henri Casanova, Arnaud Legrand, Dmitrii Zagorodnov and Francine Berman "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments" Proc. 9th Heterogeneous Computing Workshop (HCW) Cancun, Mexico 2000

[19] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on no identical processors," Journal of the ACM, Vol. 24, No. 2, Apr. 1977

[20] Howard Jay Siegel and Shoukat Ali "Techniques for Mapping Tasks to Machines in Heterogeneous Computing Systems" Purdue University School of Electrical and Computer Engineering West Lafayette, IN 47907-1285 USA June 1999

[21] Henri Casanova, Graziano Overtelli, Francine Berman and Richard Wolski "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid" Computer Science and Engineering Department University of California, San Diego USA 2000

[22] T. Yang and A. Gerasoulis "DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors", in IEEE Trans. on Parallel and Distributed Systems, vol. 5, no.9, pp.951--967, 1994

[23] S. Darbha and D.P. Agrawal "Optimal Scheduling Algorithm for Distributed Memory Machine" in IEEE Trans. On Parallel and Distributed Systems, vol. 9, no. 1, pp. 87-95, January 1998

[24] A. Radulescu and A.J.C. van Gemund, "Fast and Effective Task Scheduling in Heterogeneous Systems," The 9th Heterogeneous Computing Workshop (HCW), pp.229238, Cancun, Mexico 2000

[25] M. Wu and D. D. Gajski, "Hypertool: A programming aid for message-passing systems IEEE Trans. Parallel and Distributed Systems, vol. 1, pp. 330–343, July 1990

[26] M. Wu "MCP Revisited**"** Department of Electrical and Computer Engineering The University of New Mexico

[27] Andrei Radulescu and Arjan J.C. van Gemund "On the Complexity of List Scheduling Algorithms for Distributed-Memory Systems" Faculty of Information Technology and Systems Delft University of Technology The Netherlands 2000

[28] Andrei Radulescu and Arjan J.C. van Gemund "Fast and Effective Task Scheduling in Heterogeneous Systems" Faculty of Information Technology and Systems Delft University of Technology The Netherlands In Proceeding of Heterogeneous Computing Workshop 2000

[29] Yu-Kwong Kwok and Ishfaq Ahmad "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors." IEEE Transactions on Parallel and Distributed Systems 1996

[30] Min-You Wu and Wei Shu "On Parallelization of Static Scheduling Algorithms" IEEE Transactions On Software Engineering Vol.23 No.8 pp.517-528 1997

[31] Rajkumar Buyya and Manzur Murshed Gridsim: "A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing."

The Journal of Concurrency and Computation: Practice and Experience (CCPE), 14(13-15), 2002

[32] William H. Bell, David G. Camerona, Luigi Capozza, A. Paul Millar, Kurt Stockinger, and Floriano Zini "Optorsim - A grid simulator for studying dynamic data replication strategies." International Journal of High Performance Computing Applications, 17(4) 2003.

[33] Henri Casanova, Arnaud Legrand, and Loris Marchal "Scheduling distributed applications: the simgrid simulation framework." In Proceedings of the third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03) 2003

[34] Hiroyuki Ohsaki, Soushi Watanabe, and Makoto Imase "On Dynamic Resource management Mechanism using Control Theoretic Approach for Wide-Area Grid Computing" by in Proceedings of IEEE Conference on Control Applications (CCA 2005), 2005

[35] Eddy Caron, Vincent Garonne and Andrei Tsaregorodtsev **"**Evaluation of Meta-scheduler Architectures and Task Assignment Policies for high Throughput Computing" by Proceedings of 4th Internationnal Symposium on Parallel and Distributed Computing Job Scheduling Strategies for Parallel Processing(ISPDC'05) 2005.

[36] Y. Yang and H. Casanova "RUMR: Robust Scheduling for Divisible Workloads" by Proceedings of the 12th IEEE Symposium on High Performance and Distributed Computing (HPDC-12), Seattle, June 2003

[37] Y.K. Kwok and I. Ahmad "Benchmarking the task graph scheduling algorithms." In Proc. Int'l Parallel Processing Symp. / Symp. on Parallel and Distributed Processing, 1998.