Combining Measurement and Stochastic Modelling to Enhance Scheduling Decisions for a Parallel Mean Value Analysis Algorithm

Gagarine Yaikhom, Murray Cole, and Stephen Gilmore

School of Informatics, The University of Edinburgh,
James Clerk Maxwell Building, Mayfield Road, Edinburgh EH9 3JZ, UK
G.Yaikhom@sms.ed.ac.uk,{mic, stg}@inf.ed.ac.uk
http://groups.inf.ed.ac.uk/enhance/

Abstract. In this paper we apply the high-level modelling language PEPA to the performance analysis of a parallel program with a pipeline skeleton which computes the Mean Value Analysis (MVA) algorithm for queueing networks.

1 Introduction

From the algorithmic and performance perspectives, Grid systems are characterised by the dynamically heterogeneous nature of the resources at the programmer's disposal. This property adds a new dimension to the already challenging parallel programming task. The Enhance project [1, 2] aims to simplify the effective programming of such systems by exploiting and synthesising results from two underlying research programmes. Stochastic process algebras such as PEPA [3] are already used to model the behaviour of concurrent systems in which some aspects of behaviour are not precisely predictable. Meanwhile, skeletal parallel programming [4] recognises that many real applications draw their structure from a range of well known solution paradigms and seeks to make it easy for an application developer to tailor such a paradigm to a specific problem.

The choice of a particular skeleton carries with it considerable information about implied scheduling dependencies. In response to changing workload on servers, or unplanned unavailability due to software or hardware faults, the application can be restructured to use an alternative implementation skeleton or can simply reevaluate the parameters to the skeleton which is currently in use. Such resilience to operational faults is not typically found in low-level parallel programming approaches and is one of the strengths of a structured approach to parallel programming. Furthermore, the use of skeletons allows the programmer to provide explicit information about the future interaction structure of the application, which would be difficult or impossible to derive statically from an equivalent unstructured program source. By modelling skeletons with PEPA, and thereby being able to address aspects of uncertainty which are inherent to Grid computing, we aim to underpin systems which can make better scheduling and dynamic rescheduling decisions than less sophisticated approaches.

V.N. Alexandrov et al. (Eds.): ICCS 2006, Part II, LNCS 3992, pp. 929–936, 2006.

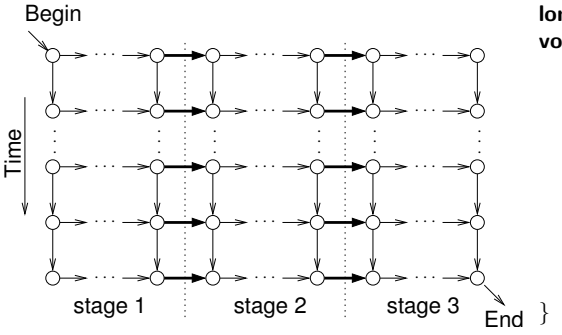
The ability to reconfigure and redeploy an application on-the-fly can be used even more effectively if the future load on the available servers can be estimated at least tolerably well. The Network Weather Service (NWS) [5] provides us with these estimates. Analogously with meteorological climate prediction, a short-range forecast is made on the basis of a record of recent activity. Perfect forecasting of future load is, of course, not achievable but in the experience of the users of the NWS, its predictions turn out to be useful more often than they are misleading and so scheduling decisions based on NWS information can be favourable ones.

In previous papers [1, 2], we have reported on development of our *pipeline* and *deal* skeletons, on their associated PEPA models, and upon the construction of AMoGeT, a modelling tool which automatically generates PEPA models for the pipelines (optionally with embedded deals), given information on the performance of the customising software components, and on resource performance obtained dynamically from NWS [5]. These PEPA models are solved using procedures of numerical linear algebra implemented in the PEPA Workbench [6].

We summarise the main technical ideas of the work here to make the present paper self-contained. Briefly, the PEPA process algebra is a timed process algebra where the duration of activities is modelled by exponentially-distributed random variables. Processes can choose between alternatives, and can be composed in parallel. Processes in a composition can be required to synchronise on shared activities. For example, a producer can enter a job into a pipeline only if the first stage is willing to accept it, and a stage in the pipeline can pass on a completed job only if a consumer is ready to receive it. Both of these are synchronisations. The slower partner in the synchronisation determines the rate of the joint activity.

Timed activities, choice, parallel composition and synchronisations are the modelling tools offered by the PEPA language. The analysis procedures are implemented in the PEPA Workbench software tool. We use PEPA models which denote finite-state stochastic (in fact, Markovian) processes. A PEPA model submitted to the Workbench will be compiled into a Continuous-Time Markov Chain (CTMC) and the long run ("steady state") probability distribution over all of the states of the model is computed using an iterative numerical procedure. From this can be calculated performance metrics such as the throughput of the pipeline, which is the basis of the comparison which allows us to rank scheduling decisions.

This paper presents the first serious application of our technology. We have taken an existing MPI program [7] for the Mean Value Analysis problem, refactored it as an instance of our pipeline skeleton (replacing the explicit MPI with our pipeline skeleton) and used AMoGeT to deploy the resulting program across a subset of the available processors of our local Beowulf cluster, while deliberately skewing the performance characteristics of the processors with a variety of artificial loads. Section 2 reviews the MVA problem and its pipelined parallelisation. Section 3 describes our experimental plan, and the expected results. Section 4 describes and discusses the actual results.



(a)

```
long int m[100]; double a[100];
void main (void) {
    int i;
    for (i = 0; i < 100; i++) {
        a[i] = (double) random();
        m[i] = random();
    }
    for (i = 0; i < 100; i++) {
        a[i] = a[100 − i] ∗ m[i];
        if (i == 99) i = 0;
    }
}
```

(b)

Fig. 1. (a) Pipeline implementation of the MVA algorithm, (b) load generation

2 The Mean Value Analysis Algorithm

The application under investigation is a numerical procedure which is used in the quantitative analysis of queueing networks. A queueing network is used to describe a system as a collection of service centres visited by customers attempting to fulfil their service requirements. The customers under study may very often be jobs executing on a variety of servers. The objective is to determine meaningful performance metrics about the system such as average queue length and utilisation of servers. This information can be used to rectify performance problems which stem from the design of the network. Typically this will involve network re-structuring or the introduction of additional servers in order to decrease mean queue length and increase utilisation.

Three distinct types of analysis can be applied to queueing networks, depending on their structure and the service discipline used at service centres. Firstly, closed-form analytic solutions exist for open queueing networks for a limited class of service disciplines if customers do not re-enter the network after their service is complete. Secondly, numerical algorithms may be applied to open or closed product-form networks with Poisson arrivals and exponentially-distributed service times with the possibility for customers to loop back and re-enter the system. Finally, Monte Carlo discrete-event simulation may be applied in other cases. These three types of analysis have increasing evaluation cost. We are considering the second type here.

Numerical algorithms compute the probability of a network of M service centres being in a state $\mathbf{n} = (n_1, n_2, \ldots, n_M)$ using a descriptor of the form

$$\Pr(\mathbf{n}) = \frac{1}{G} \prod_{i=1}^{M} f_i(n_i) \tag{1}$$

where G is a normalisation constant for the probability values and f is a function which depends upon the type of the service centre [8].

The Mean Value Analysis (MVA) algorithm [9] computes performance metrics such as utilisation and average queue length for such queueing networks

using increasing customer populations, from 0 to N. In the present study we base our skeletal program on an existing parallel implementation of MVA due to Gennaro and King [7]. Their implementation uses MPI to coordinate the computation and has the structure of a pipeline, with task dependencies as shown in Figure 1(a). We have taken the computational cores of their code, and embedded them in a skeleton program, replacing the explicit use of MPI with a call to our pipeline skeleton (shown in Figure 2(a)). The MVA application is novel in that the number of stages in the pipeline is not an implicit property of the algorithm, but is dictated by the granularity of the partitioning of the implicit task graph (Figure 1(a)). As illustrated, we have arbitrarily chosen a three stage version. Changing the granularity is trivial (redefining a constant) in both original MPI and skeletal versions of the code.

3 Experimental Plan

Our immediate goal is to demonstrate experimentally that AMoGeT can use its PEPA pipeline model, informed by dynamically gathered resource performance information, to choose a good allocation of pipeline stages to processors. To this end, we have chosen to work with a three stage version of the MVA pipeline, and a five processor subset of a Beowulf cluster. These numbers are small enough to allow us to ask AMoGeT to generate and evaluate all possible schedules (which three processors to use, and in which order). As the implied search space grows exponentially, exhaustive search will eventually become prohibitive. Pruning the search space is an orthogonal challenge. The work reported here demonstrates that AMoGeT provides a sound objective function upon which to base heuristics.

Our experiments are grouped into sets of six runs. In each set, we artificially create situations in which zero, one, two, three, four and five of the available processors are seeded with heavy computational loads. Without loss of generality, and to assist the reader, we choose processors with lowest index (or, rank) when loading processors. For example, to load three processors we choose processors 1, 2, and 3. Since our Beowulf currently has a "free-for-all" allocation policy, there may be other sporadic communication and computation loads placed by other users.

For each run, in each set, we ask AMoGeT to gather information and predict which three of the five available processors will provide the best and worst overall application performance (remembering that in a pipeline, this choice should be affected by both computation and communication resource availability). We then run (in sequence) these two schedules, noting the execution times on each of the selected processors.

For each set of experiments, we present a total of 36 measured run times. These are grouped into six pairs of three measurements (each measurement corresponding to a pipeline stage). Each pair corresponds to experiments with a fixed number of "bad" processors, grouped left to right in the graph from zero bad processors to five bad processors. Within each pair, to the left we find the real run times on the three processors in the predicted best case, and to the

right the real run times on the three processors in the predicted worst case. For example, in Figure 2(b), the second pair of runs (indexed (2,5,4) (5,1,3)) indicate that it was predicted that using processors 2, 5 and 4 would give the best performance and that using processors 5, 1 and 3 would give the worst performance. The height of the bars gives the actual run time on each of these processors in the corresponding run, and the figure of 63.29% at the top indicates the improvement from worst run to best run.

We present results for two such sets of experiments. In the first set, conducted on a cluster with low background interference from other users, our "bad" processors were artificially loaded to have around 33% of the availability of the "good" processors. In the second set, conducted on a cluster with higher background interference from other users, the "bad" processors' availability is improved to around 66%.

An intuitive assessment of the situation, based on conventional wisdom about pipeline performance, might suggest the following outcomes:

- we should choose three "good" processors, when these are available;
- the gap between best and worst case schedules should be widest in those situations for which it is possible to choose all "good" processors in the best case, and at least one "bad" processor in the worst case (which for our set-up occurs only in the "one bad processor" and "two bad processors" runs);
- both of the preceding observations may be tempered for situations in which the performance of computationally "good" processors is hampered by excessively bad communications performance to intended neighbours.

4 Experimental Outcomes

The experiments were conducted on two separate Beowulf clusters (16 nodes and 64 nodes) with RedHat Linux Fedora Core 3 release, version 2.6.12. We used LAM-MPI implementation of the MPI standard, version 7.1.1. For extracting CPU availability per node, and inter-node TCP communication latency, we used the Network Weather Service (NWS), version 2.13.

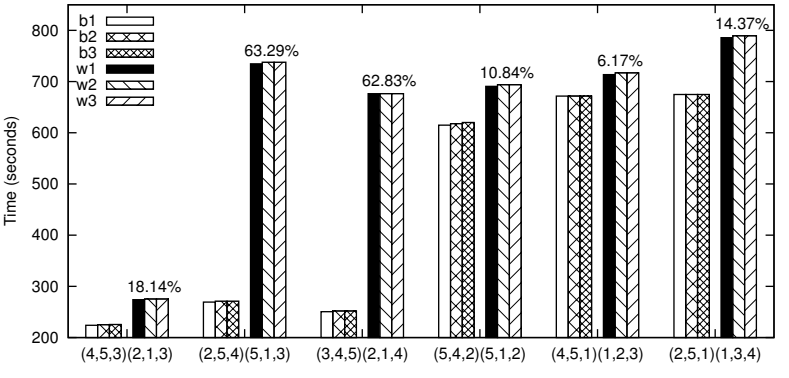
The number of classes in the queueing network is set to three, with each class having a population of 1024. The number of resources in the queueing network is set to two resource queues. For the artificial load simulation, we use the program shown in Figure 1(b), adapted from [10, page 171].

The experimental results are shown in Figure 2. The results show that the mapping suggested by our approach produces satisfactory improvements, as expected. The results for the cluster with low background interference and CPU availability reduced to 33%, show improvements of up to 63% when stages can be mapped to unloaded nodes. This choice is reflected in the corresponding mapping. When more than two nodes are loaded, our model still chooses the best possible mapping, producing an improvement of around 10%. The results for the cluster with more background interference and CPU availability reduced to 66%, show improvements of up to 29%. Because of the sporadic interference, the

(a)

```
void first_stage (void) {
  do { /* Perform computations and update queue lengths. */
    if (node_idx[0] == end) PipeGive (tmp_qlen, dtype, 1); /* Send queue lengths. */
  } while (update_node_idx()); /* Continue if not done. */ }
void last_stage (void) {
  do { if (node_idx[0] == start) PipeTake (tmp_qlen, dtype, 1); /* Receive queue lengths. */
    /* Perform computations and updates. */ } while (update_node_idx());
  /* Generate output. */ }
void inter_stage (void) {
  do {/* Receive, compute, update and send queue lengths. */} while (update_node_idx()); }
int main (int argc, char *argv[]) {
  SkelLibInit(); /* Initialise skeleton library. */
  for (i = 0; i < NSTAGES+1; i++) {spreads[i] = SPLOCAL; types[i] = dtype;} /* Data map. */
  for (i = 0; i < NSTAGES; i++) amodes[i] = SKEL; /* Stage type. */
  mystage = myrank(); /* Process to stage mapping. */
  stages[0] = first_stage; stages[NSTAGES-1] = last_stage; /* Assign stage function. */
  for (i = 1; i < NSTAGES - 1; i++) stages[i] = inter_stage; /* Assign stage function. */
  Pipeline(NSTAGES, amodes, stages, mystage, BUF, spreads, types,
      NULL, 0, 0, NULL, 0, &outmul, 0, mycomm()); /* Execute pipeline. */
  /* Output results, and finalise. */
}
```

(b)



(c)

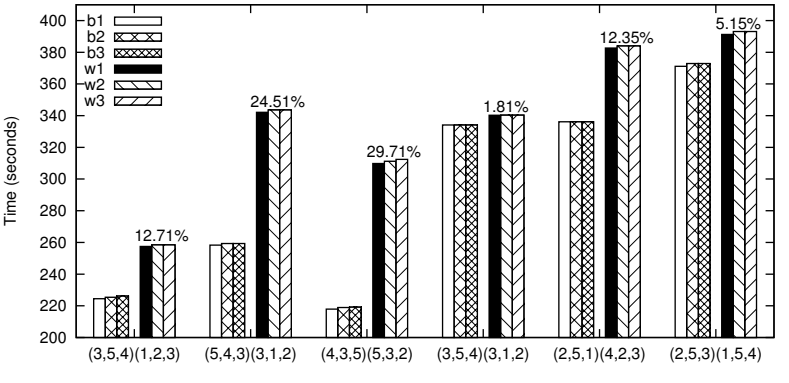


Fig. 2. (a) Implementation source code (simplified), (b) 33 percent availability (16 node cluster with low background interference), (c) 66 percent availability (64 node cluster with higher background interference). b_i and w_i respectively correspond to the best and worst stage i mapping.

performance improvements display some variation, giving an average improvement of about 8% when all nodes are unloaded, or one or all of the stages are mapped to loaded nodes. Considering the average execution time of 40 seconds to generate and solve the pipeline PEPA models for the 60 possible mappings, our approach proves to be pragmatic and advantageous.

In contrast to our use of computational load generation, we made no attempt to impose artificial communication load. Nevertheless, detailed inspection of the NWS readings (not reproduced here) reveals that real imbalances in communication load arose from the background on a number of occasions, sometimes with interesting effects on predicted best and worst schedules. For example, in Figure 2(c), in the two loaded processor case (these being processors 1 and 2, as explained above), there is an unusual spike in predicted latency between processor 5 and processor 3. The predicted worst case schedule (5,3,2) chooses to exploit this weakness, even at the expense of not choosing the weak processor 1. Similarly, in the (rightmost) "all" loaded case, there is a very high spike in the latency between processors 1 and 5. Again this is exploited in the worst case and avoided in the best, although the difference in actual run times between these cases is smaller than might have been expected. It may be that this variation is attributable to the inevitable distinction between predicted latencies, as used in scheduling, and actual latencies experienced at run time.

5 Conclusions

In this paper we have applied performance analysis methods based on stochastic process algebra to an exemplar of the class of structured parallel programs with a pipeline configuration of tasks. Our exemplar is the parallel implementation of the MVA algorithm for queueing networks. Our methods are general ones: we are using MVA simply as an example pipeline-structured application. Thus we have not made alterations to our methods to take advantage of specific features of MVA which are not usually found in a pipeline-structured parallel program.

By basing our performance predictions on measurements of network latency and compute load we are able to predict ahead-of-run those scheduling decisions which will lead to the shortest run-time and those which lead to the longest run-time. In the present study we did not find a single instance where the performance predictions computed by the PEPA Workbench were misleading: predicted best cases always significantly outperformed predicted worst cases in practice. Our analysis is fully automatic and runs without user intervention. The analysis cost is modest. This is essential for maintaining the freshness of the measurement data and so that the scheduling overhead does not impede the progress of the main computation. The relative speed of the analysis points towards its suitability for use in on-line scheduling decisions.

In the present paper our dynamic scheduling is performed *before run*. Our eventual plan is to allow dynamic rescheduling *within a run* so that downstream computations benefit from measurements obtained during the upstream computation.

6 Related Work

The combination of stochastic modelling, structured parallelism and dynamically derived performance monitoring used to inform our scheduling approach is, we believe, unique. Previous skeletal and similarly structured systems have used analytical performance models to guide scheduling on homogenous resources [11, 12, 13], and information gathered from NWS to guide scheduling in heterogenous contexts [14].

Acknowledgements. The authors thank Jane Hillston for helpful discussions and Horacio Gonzalez-Velez for helpful suggestions on the usage of NWS. The Enhance project is funded by the UK EPSRC grant GR/S21717/01.

References

1. Benoit, A., Cole, M., Gilmore, S., Hillston, J.: Evaluating the performance of skeleton-based high level parallel programs. In: ICCS 2004: Proceedings of the 4th International Conference on Computational Science, Kraków, Poland. Volume 3038 of LNCS., Springer-Verlag (2004) 289–296
2. Benoit, A., Cole, M., Gilmore, S., Hillston, J.: Scheduling skeleton-based grid applications using PEPA and NWS. The Computer Journal 48(3):369-378, 2005. **48** (2005) 369–378
3. Hillston, J.: A Compositional Approach to Performance Modelling. CUP (1996)
4. Cole, M.: Algorithmic Skeletons: Structured Management of Parallel Computation. MIT Press & Pitman (1989)
5. Wolski, R., Spring, N., Hayes, J.: The Network Weather Service: a distributed resource performance forecasting service for metacomputing. Future Generation Computer Systems **15** (1999) 757–768
6. Gilmore, S., Hillston, J.: The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In: Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation. Volume 794 of LNCS., Vienna (1994) 353–368
7. Gennaro, C., King, P.: Parallelising the Mean Value Analysis Algorithm. Transactions of The Society for Computer Systems International **16** (1999) 16–22
8. Baskett, F., Chandy, K., Muntz, R., Palacios, F.: Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. JACM **22** (1975) 248–260
9. Reiser, M., Lavenberg, S.: Mean Value Analysis of Closed Multichain Queueing Networks. JACM **27** (1980) 313–322
10. Gunther, N.J.: Analyzing Computer System Performance with Perl::PDQ. Springer (2005)
11. Scaife, N., Horiguchi, S., Michaelson, G., Bristow, P.: A parallel SML compiler based on algorithmic skeletons. Journal of Functional Programming **15** (2005) 615–650
12. Morajko, A., Cesar, E., Caymes-Scutari, P., Margalef, T., Sorribes, J., Luque, E.: Automatic Tuning of Master Worker Applications. In: Proceedings of Euro-Par 05. Volume 3648 of LNCS., Springer-Verlag (2005) 95–103
13. Cosmo, R.D., Li, Z., Pelagatti, S., Weis, P.: Skeletal parallel programming with ocamlp3l 2.0. In: To appear in Parallel Processing Letters. (2006)
14. Gonzalez-Velez, H.: An Adaptive Skeletal Task Farm for Grids. In: Proceedings of Euro-Par 05. Volume 3648 of LNCS., Springer-Verlag (2005) 401–410