Online Scheduling in Distributed Message Converter Systems

Thomas Risse, Andreas Wombacher
GMD – IPSI, Integrated Publication and Information Systems Institute,
Dolivostraße 15,
64293 Darmstadt, Germany
{risse, wombach}@darmstadt.gmd.de

Mike Surridge, Steve Taylor
IT Innovation Centre, 2 Venture Road, Chilworth Science Park, Southampton
SO16 7NP, UK
{ms,sjt}@it-innovation.soton.ac.uk

Karl Aberer
LSIR - Distributed Information Systems Laboratory
EPFL-DSC, CH-1015 Lausanne, Switzerland
karl.aberer@epfl.ch

Abstract

The optimal distribution of jobs among hosts in distributed environments is an important factor to achieve high performa nce. The optimal strategy depends on the application. In this paper we present a new online scheduling strategy for distributed EDI converter system. The strategy is based on the Bin-Stretching approach. The original algorithm has been enhanced to satisfy the business goals of meeting deadlines, priority processing, low response time and high throughput. The algorithm can be flexible adapted to different objective goals due to its two-stage strategy. We show by simulation and measurements on a real system that the modified Bin-Stretching approach fulfills the objective goals while requiring only low computational effort.

Keywords: Online scheduling, load balancing, distributed message conversion, performance measurement

1. Introduction

Electronic data interchange is an important part of the implementation of business processes. The exchange of data between heterogeneous systems requires support for different data formats (EDIFACT, XML, etc.). Enterprises use different proprietary in house formats. So the incoming and outgoing messages must be converted from the inbound format to the in-house format, as well as from the in-house format to the outbound format. The volume of data each enterprise delivers and receives will grow rapidly in the next years. This leads to growing demands on performance of EDI converter systems.

This problem was the motivation to investigate in the project POEM (Parallel Processing Of Voluminous EDIFACT Documents) the question building performant parallel converter system, based on the typical infrastructures currently available in large enterprises. Possible system configurations range from one-processor machines to clusters of different machines with different performance characteristics. This requires a flexible and effective scheduling mechanism for distributing the incoming messages on the available computing resources.

1.1 Overview of the paper

In Section 2 we give more details on the application we are addressing and we will state the scheduling problem. In Section 3 an overview of existing approaches to scheduling is given and the Bin-Stretching strategy underlying our approach to scheduling is introduced. Section 4 describes the scheduling strategy fo r the converter system and introduces the *modified Bin-Streching* scheduling algorithm. The algorithm is analysed and compared to other algorithms in Section 5. In Section 6 we report on the measurements that were performed for the POEM system and which confirm the theoretically obtained results. Finally Section 7 gives conclusions and an outlook on future work.

2. Application Background

In the banking sector large EDI messages containing transaction information need to be converted from an inbound format to the formats of in-house systems. The goal of the system architecture is to achieve reliability, scalability and high throughput.

To satisfy these requirements a parallel architecture is used. The system can be built from different host types, allowing the use of existing hardware and the incremental extension of the system with new hardware. The generic architecture of the system is shown in Figure 1. Since the system architecture is based both on the use of distributed machines and parallel machines the scheduling has to be performed at two different levels.

The *global scheduling* is the first level. At this level the incoming jobs are distributed to the available machines considering the following goals and constraints:

- Meeting of deadlines: All messages which arrive within a given arrival time will be processed till the next deadline.
- Priority messages: EDIFACT messages can have a priority flag. The processing of these messages must start within a short time range.

- Low response times: The total time to process a message must be as low as possible.
- High throughput: The number of processed messages must be as high as possible.

In our processing model a job is defined as the conversion of a part or of a complete message, as shown in Figure 1. So it is possible to distribute the conversion of a message over several hosts. In our system model a job can only be assigned to one machine to avoid moving processing information and data among the machines as this is prohibitively expensive. The only exception to that is a machine failure.

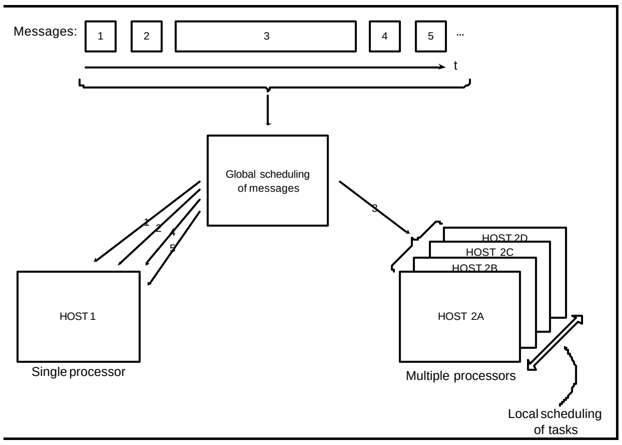

Figure 1: System architecture

A job should normally not be preempted because this requires too much control and communication overhead between the scheduling levels. The only exception we consider is the arrival of a message with priority flag. In this special case the processing must start with as little delay as possible.

In the describ ed scenario the problems (jobs) arrive at an unknown future time. So scheduling has to be done with the partial knowledge available in runtime. Therefore we have an *online scheduling* problem. The schedules have to be updated every time a new message arrives.

The *local scheduler* controls the execution of tasks and the distribution of the tasks to the processors on the different hosts. The local scheduler is tightly coupled with the operating system. Hence in the following we are focusing on the scheduling of jobs at the global level.

A more detailed description of the architecture and the processing steps can be found in [13].

3. Selection of a Scheduling Strategy

The selection of an appropriate scheduling strategy is important for the performance of the EDI converter system.

Scheduling algorithms and load balancing algorithms are extensively studied for different applications. General overviews regarding scheduling are given, e.g., in [5] and [7] and for load balancing, e.g., in [1] and [17]. Even if both research areas are overlapping in the following we put our focus on scheduling. Load balancing puts the main emphasis on the goal of achieving balanced system load rather than optimized response times and throughput.

Online scheduling algorithms have been thoroughly analyzed in the literature, see e.g. [15]. Non-preemptive online scheduling algorithms have first been evaluated by Graham [10]. His list scheduling algorithms have been improved in [4][9] to give better results for problems with large number of machines. An alternative approach has been taken by Albers [1], who uses a load threshold for scheduling decisions. A similar strategy has been developed by Azar et al.. His *Online Bin-Stretching* strategy [3] will be the starting point for our algorithm and is described and extended in Section 4.

The objective function for most of the online algorithms is to shorten the makespan. Other goals like deadlines or response time are less frequently considered. For those objective functions mostly list scheduling algorithms are adapted, e.g. Shortest Deadline First (SDF) [5][7]. Scheduling with deadline constraints is also considered for real time systems. But these systems often have periodic tasks with precedence constraints [6][8].

Results for scheduling in specific application areas are most frequently found in the areas of operating systems (e.g. [16][14]) and production systems (e.g. [5]). Those application-specific algorithms are not applicable for our scheduling problem in a distributed message converter system as their goals and constraints are very application specific.

None of the algorithms we looked at could satisfy the requirements that we had identified for the scheduling problem that we have identified for the EDI converter system directly. Therefore we had to develop a new scheduling algorithm. For that, we were using the Online Bin-Stretching strategy as point of departure, because from [3] it is know that the strategy gives short makespans independent of the number of machines. In addition it requires only little computational power and it can be easily adapted to different goals.

4. Scheduling Strategy

The scheduling problem for distributed EDI converter system has to be partitioned into several sub problems, for which different scheduling strategies apply.

Figure 2 shows the decision tree for the global scheduler.

After arrival of a new job the global scheduler selects a suitable scheduling strategy depending on the priority of a job. All other jobs go to the normal processing. A modified version of Bin-Stretching [3] is used for the nomal processing. Its two stage scheduling strategy selects first an appropriate host for the job. In the second step the jobs are sorted on the assigned host by a list scheduling algorithm (s.a. [5][10]), e.g. FCFS, SDF.

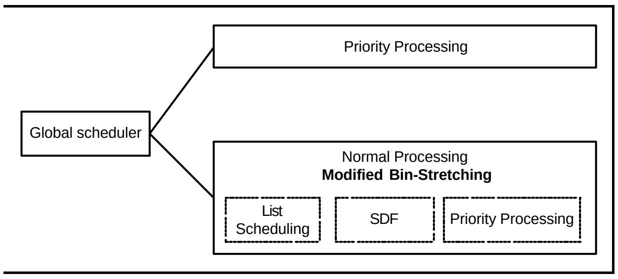

Figure 2: Scheduling sub problems

4.1 Online Bin-Stretching

The idea of the original ‘Online Bin-Stretching’ algorithm presented in [3] is to pack a sequence of items into a fixed number of bins. Each bin represents one host. Bin-Stretching is somewhat related to the bin-packing problem. In both cases all the items are to be packed in bins of a certain size. In bin-packing the goal is to minimize the number of bins where the bin size is fixed while in Bin-Stretching the number of bins is fixed and the goal is to minimize the stretching factor of the bins. The original algorithm assumes that the optimal job load of a bin, the sum of the processing times of all waiting jobs in that bin, is known in advance. As of the dynamic nature of the scheduling problem that additional information is not available. Other restrictions of the original algorithm are that it does not consider other goals like deadlines and that it does not allow the changing of the execution order and that the hosts must be identical.

4.2 Modified Bin-Stretching

We describe now the modified Bin-Stretching algorithm that we are using. The extensions concern the following points:

- The optimal load of the host (=bin) must not be known in advance.
- The execution order of jobs can be changed after scheduling.
- Additional goals are taken into account, e.g. deadlines and priorities.
- Non-identical hosts are considered.

The modified Bin-Stretching algorithm assigns a job in two steps. The first step is to select an appropriate host such that the load of all hosts is balanced. The second step is to reorder the execution queue of the host. Hence it is possible to consider secondary goals, e.g. the deadlines. This can be done by specifying a priority order on the execution queue of each host. A priority order for the global scheduler is in decreasing priority as follows:

1. Priority jobs

2. Suspended jobs which have to meet the next deadline
3. Pending jobs which have to meet the next deadline
4. All other suspended jobs
5. All other pending jobs

To describe the modified Bin-Stretching algorithm we have to introduce first some definitions.

Hosts

M={ $m_1, \ldots, m_k$} with s_1 ³ s_2 ³ … ³ s_k ; *k*=number of hosts

s_l : speed factor of host m_l (usually compared to the slowest host); *l=1 ,…, k*

Jobs

j=(f,e): *j* defines a job with the following properties:

f : estimated flow time

e : execution flag

e=0 : job is not executed

e>0 : the job is currently executed or the execution is finished

f(j) : returns the estimated flow time of job *j* measured on a reference host

e(j) : returns the execution flag of job *j*.

rt(j,t): remaining runtime time of job *j* at time *t* measured on a reference host

$$rt(j,t) = \begin{cases} f(j) & \text{if the execution of job j is not started} \\ \text{Remaining runtime} & \text{if the execution of job j is started and not finished} \\ 0 & \text{otherwise} \end{cases}$$

J={ $j_1, \ldots, j_n$ } ; *n*=number of jobs; the indices of the jobs reflect the order in which they arrived at the system.

J_l : All jobs assigned to host m_l ; *l=1 ,…, k*

In addition miscellaneous functions are required. In the following definitions *p* with *1£p£n* represents always an index to an individual job in *J* or J_l with *l=1 ,…, k* .

Maximum flow time of the jobs $j_1 \ldots j_p$:

$$F_{\max}(p) = \max_{i \le p}\{ f(j_i)\} \qquad (4.1)$$

Maximum flow time of the jobs j_i Î J_l with *i£p*:

$$F_l\ (p) = \max_{i \le p \wedge j_i \in J_l} \{ f(j_i)\} \qquad (4.2)$$

Job load of host m_l at time *t* with j_i Î J_l and *i£p*:

$$L_l(p,t) = \sum_{i \le p \wedge j_i \in J_l} \frac{rt(j_i,t)}{s_l} \qquad (4.3)$$

Maximum load of all hosts in *M*:

$$L_{\max}(p) = \max_{l \le k}\{L_l(p)\} \qquad (4.4)$$

A host is said to be short if its load is at most $a \cdot L_{\max}(p)$. Otherwise, it is tall. The value of *a* can range between 0 and 1. It determines the threshold among a short and tall host. The value of *a* influences the quality of the resulting schedule. An analysis regarding the optimal value of *a* can be found in Section 5.

When a job j_p arrives the disjoint sets S1, S2 and S3 are defined as

$$S_1 = \left\{ 1 \le l \le k \mid L_l(p-1) + \frac{f(j_p)}{s_l} \le a \cdot L_{\max}(p-1) \right\} \quad (4.5)$$

$$S_2 = \left\{ \begin{array}{l} 1 \le l \le k \mid L_l(p-1) \le a \cdot L_{\max}(p-1), \\ a \cdot L_{\max}(p-1) < L_l(p-1) + \frac{f(j_p)}{s_l} \\ \le a \cdot L_{\max}(p-1) + \frac{F_{\max}(p-1)}{s_l} \end{array} \right\} \quad (4.6)$$

$$S_3 = M \setminus (S_1 \cup S_2) \quad (4.7)$$

S_1 contains all hosts that job loads are short or remain short if the current job is placed on them. S_2 is a set of hosts that job loads are short but become tall if the job is placed on them. The job load of the hosts in S_2 is stretched at most to $a \cdot L_{\max}(p-1) + \frac{F_{\max}(p-1)}{s_l}$. S_3 contains all hosts which current job load is over $a \cdot L_{\max}(p-1)$ or needs to be stretched over $a \cdot L_{\max}(p-1) + \frac{F_{\max}(p-1)}{s_l}$ if the job is placed on them.

Examples for the different sets are shown in Figure 3.

In the next step of the algorithm a host must be selected from the three sets.

- Put the job on the currently fastest non-empty host from the set S_1. If S_1 contains only empty hosts then put the job on the currently fastest empty host.
- If S_1 = Æ then put the job on the least loaded machine after assignment ($= \min_{l \in S_2} \left\{ L_l(p-1) + \frac{f(j_p)}{s_l} \right\}$) from the set S_2.
- If S_1 = S_2 = Æ then put the job on the least loaded machine after assignment ($= \min_{l \in S_3} \left\{ L_l(p-1) + \frac{f(j_p)}{s_l} \right\}$) from the set S_3.

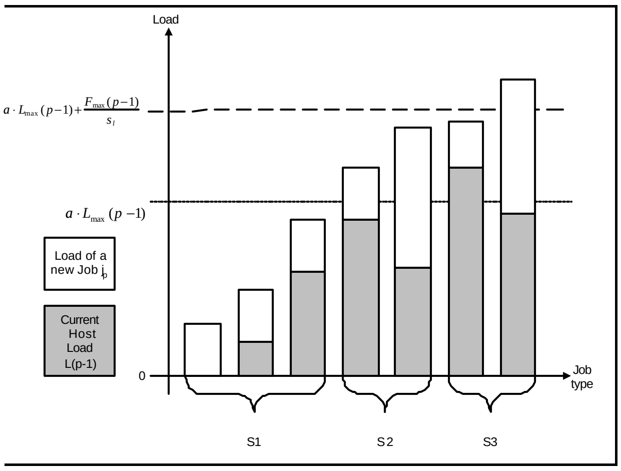

Figure 3: Examples for the sets S_1, S_2 and S_3

After assigning the new job j_p to the selected host, the job is stored in the waiting queue of the host. The queues are ordered according to the sorting criteria given before.

The secondary goals are not optimally achieved from the global viewpoint, because they are only considered at the individual hosts. In the worst case it is possible, that, for example, all jobs for the next deadline are scheduled to one host and the other hosts contain only jobs for later deadlines. If such case occurs the algorithm can be extended in a way that it checks at the cut-of-time if there exists any host, which can keep the next deadline. Afterwards the jobs, which will miss the deadline, are moved to hosts where they can meet the deadline.

4.3 Priority Processing

The goal for the processing of priority jobs is to start the processing with a short delay and keep the response time as short as possible. When a new priority job arrives the strategy is as follows:

- If one or more underloaded hosts are available, assign the job to the fastest one.
- If no underloaded host is available, select the fastest host, which is not processing a priority job. One of the running jobs on that host must be suspended.
- If all hosts are processing priority jobs, then put the job into the waiting queue. It can be scheduled like any other job with the Bin-Stretching algorithm described in Section 4.2. The ordering of the waiting queue ensures that all priority jobs are processed first.

This strategy ignores the other scheduling strategies and unbalances the system. So we have to reschedule the pending jobs. The following possibilities exists:

- No rescheduling: If the priority jobs were rather small compared to the other jobs the unbalances are also be very small. This will be compensated after the arrival of the next jobs
- Rescheduling of a similar job: The rescheduling is done for a job, similar to the priority job on the selected host. This requires only one rescheduling operation. The result can be suboptimal because it is not always possible to find a similar job.
- Rescheduling of all jobs: This can be time intensive but gives better result.

To achieve a good trade off among effort and optimality, the combination of all three possibilities can be used. If the priority jobs are very small then no rescheduling will be done. For huge jobs the rescheduling of a similar job is suitable. Only if no similar job exists a complete rescheduling of all jobs is performed.

5. Analysis

In this Section the modified Bin-Stretching algorithm is analyzed with respect to its behavior for different scenarios corresponding to different message distributions. The simulations used for the analysis have been performed with the scheduling simulator introduced in [13]. It can simulate different scheduling algorithms on different host configurations. It can use different strategies for generating jobs, e.g. random or using realistic distributions. The generated jobs can also be stored for repeating tests with the same data.

The test scenario used consists of 95% small messages with 38 sec average processing time and 5% large messages with 3459 sec average processing time. The expected processing times are based on measurement of a real system with a single processor (s.a. Section 6). The test scenario represents a typical message distribution for the processing of EDI messages in the banking sector.

5.1 Simulation results

The first analysis refines the initial results reported in [13]. There we have shown that the algorithm gives good results in the two machine case with a six and two processor host. The second analysis complements these results with an additional host configuration consisting of two hosts with six processors and one host with two processors. The modified Bin-Stretching algorithm will be compared to the ‘First Come First Serve’ strategy (FCFS). The test scenarios assume a fully loaded system. Hence non-optimal schedules increase the number jobs, which miss the deadline.

The charts in Figure 4 and Figure 5 show histograms of the differences in the response time and lateness of jobs between modified Bin-Stretching and FCFS. Each bar represents the sum of job complexities belonging to one lateness class. Hence large jobs have more impact on the bar size then small jobs. This reflects the importance of large messages over small messages for the banks. Positive values indicate better performance for the Bin-Stretching strategy, while negative values indicate better performance for FCFS. The chart also contains a comparison for different α values.

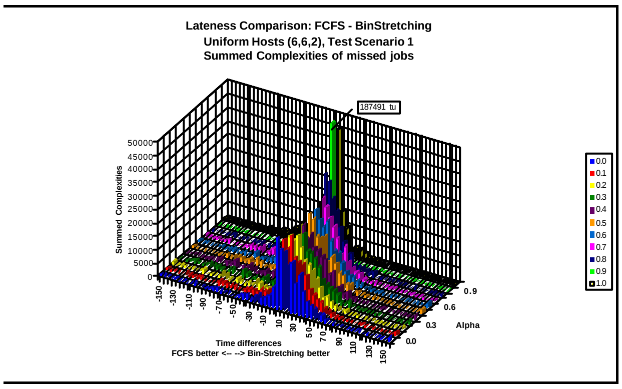

Figure 4: Response time comparison

With Bin-Stretching most jobs have the same or shorter response times than with FCFS. This can be seen in Figure 4. It can also be seen that the correct selection of the threshold value α is important for the algorithm. For large values the algorithm behaves more like FCFS. Small values give better results. Hence in Section 5.2 a method is described to determine the correct threshold. We also analyzed how the jobs keep the deadline and found similarly positive results.

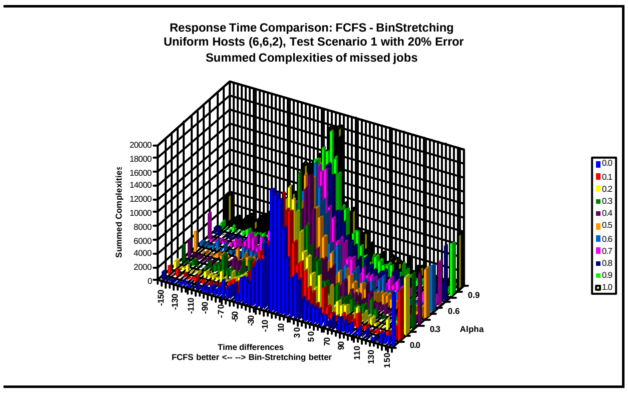

Figure 5: Job lateness with 20% proc. time variance

As the algorithm requires an estimation of the processing time it is important to know how it reacts if the real processing times deviate from the estimated times. Measurements on the real message converter system have shown that the processing time estimation can differ from the real processing time up to 20%. Hence for the next simulations we vary the real processing time in the range of ±20%. Figure 5 shows the result. It can be seen that the differences are wider spread but the positive effect of Bin-Stretching remains the same. The peaks at the extreme ratios indicate that there exist much more jobs outside the examined range. This also shows the benefit of the algorithm because these peaks are rather high compared to the previous case with no estimation error.

5.2 Selection of the optimal threshold

Next we performed simulations to find the right threshold α for the modified Bin-Stretching algorithm. The test scenario used is the same as in the previous analysis.

As one important goal of the scheduler is to keep the deadlines this objective is used to select the optimal threshold. The scenario is executed with several threshold values. The results can be seen in Figure 6.

The figure shows the summed processing times of all jobs, which missed the deadline. The simulations have been done with a processing time variation of 20%. In addition the same simulations have been done for the FCFS strategy. The modified Bin-Stretching performs up to 10% better for threshold values $0.1 < \alpha < 0.4$. As the value of 0.5 is to near the upper bound and the good result can be an exception from the general trend we select 0.3 as the threshold value.

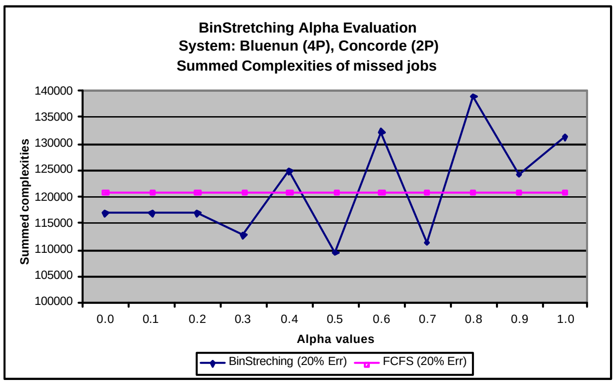

Figure 6: Summed processing tines of late jobs

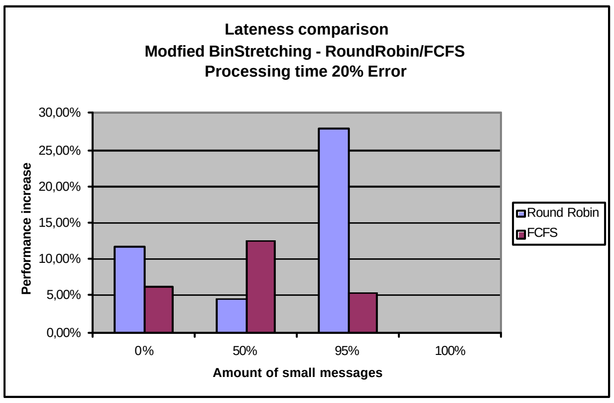

Figure 7: Lateness for different message distributions

The good performance of Bin-stretching remains also if the message distribution is varied. The results are shown in Figure 7. It shows the benefit for the modified Bin-Stretching compared to FCFS and Round Robin. The simulations have been done again with a processing time error of 20%. The results for the modified Bin-Stretching are in minimum 5% better for all messages distribution except for 100% small messages. In this case all algorithms behave in the same way because only small jobs have to be distributed which gives the greatest flexibility for the distribution.

6. Performance measurements

This section describes the performance testing on a real system at IT Innovation Centre, which provides independent verification of the theoretical analysis of the modified bin-stretching scheduler.

Testing has been conducted to address a number of objectives. These are listed below.

- To find the maximum saturated throughput of the system to give a useful guide to the system's performance.
- To investigate the sensitivity of the system to variation of the message population.
- To investigate the sensitivity of the system to different scheduling algorithms, and to verify that the modified Bin-Stretching algorithm produces the optimal schedule.

To this end, a number of different tests using different test data and system configurations were conducted. A number of parameters were varied, mainly: scheduling algorithm and message distribution. Other parameters, such as software and hardware configuration were varied to give dimension and a reference point for the results.

Two hardware systems were used. These are listed below.

- An IBM F50 host (named "bluenun"). This is a 4-processor machine and has the POEM system and the database installed, both locally.

- A cluster of a 4-processor IBM F50 (“bluenun”) and a 2-processor IBM F50 (“concorde”). This is configured as a master-slave arrangement The master (Bluenun) has the database and the server components of the POEM system.

6.1 Single Host Tests

This used the host ‘bluenun’ and aimed to investigate the system’s response to different message distributions to give a baseline to compare multiple-host results against. The message distribution was varied by changing the ratio of small messages to large messages. Hence the following message populations are used:

- 0% small messages, 100% large messages – 0 small messages, 5 large messages;
- 50% small messages, 100% large messages – 5 small messages, 5 large messages;
- 95% small messages, 5% large messages – 95 small messages, 5 large messages;
- 100% small messages 0% large messages – 95 small messages, 0 large messages.

The measurements show that the highest performance is where there is the greatest amount of work. This is in the case where there are no small messages. For a single host machine, large packets of work are the most efficient. The results for processing are as follows:

Table 1: Single host results

Run	# PAYMULS	Test	Trans / Hour
1	3	0% small 100% large	193133
1	3	50% small 50% large	192885
1	3	95% small 5% large	173595
1	3	100% small 0% large	57000

6.2 Distributed System Tests

The tests in this Section investigate the distribution of processing on a heterogeneous cluster. This is the “bluenun and concorde” system referred to previously. The main aims of the tests were to verify the scheduling algorithms, and to find the maximum throughput of the system as a whole. Messages were distributed as described in the previous section.

All available scheduling algorithms were tested. These are listed below.

- Round Robin – simple distribution based on a circular pattern.
- First Come First Served – first available host gets the next scheduled item of work.
- Random (also known as fuzzy) – random distribution of work.
- Bin-Stretching - the modified Bin-Stretching algorithm described in this paper.

The test sets described in Section 6.1 were applied to the system using each scheduling algorithm, and the throughput of the system was measured. A summary of all results is shown in Figure 8.

The round robin scheduler was tested first. This was not expected to be optimal because it ignores the relative performance of the two hosts. The maximum throughput attainable using this scheduling algorithm is 225K transactions / hour. This is for the case where there are no small files. This is to be expected since the system is highly loaded. However, the greatest performance increase over a single host occurs when there are 95% small files and 5% large files, even though this does not produce the greatest throughput. This is mainly because the small files may be processed by two packers (the bottleneck) and ‘fill in gaps’ of processing where on a single host system there are none.

The results for the FCFS scheduler show that there is a considerable improvement using this algorithm compared to the round robin algorithm. Here, a maximum throughput of 264K transactions / hour is attainable.

Results using the fuzzy (random) scheduling algorithm were of slightly lower performance than the FCFS results. This scheduler is not expected to be optimal by its random nature and the nature of the test data. Were the test data to have a random distribution over an infinite time, then the random nature of the scheduler may suit it better. Given the coarse grain of the test data used, the heterogeneous hardware platform and the finite time of the test, it was not expected to perform well.

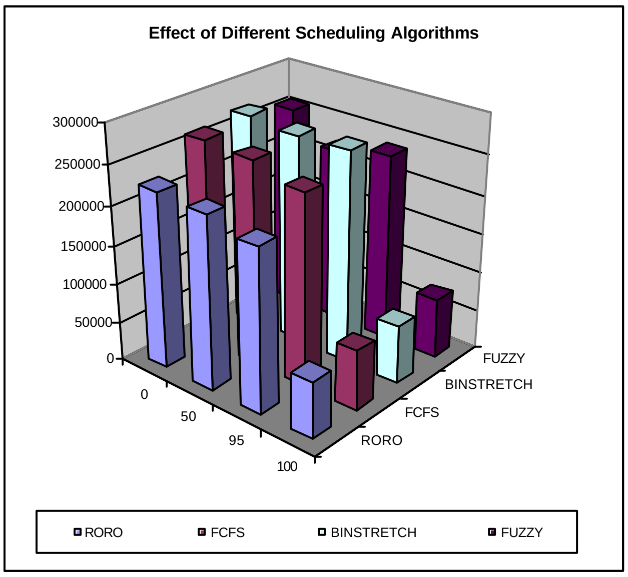

Figure 8: Effect of Different Scheduling Algorithms

The modified bin-stretching scheduler shows a significant performance gain, and the overall throughput were greater than that demonstrated by the FCFS algorithm. The maximum throughput was 272K trans / hour, where there were no small files.

The conclusion of these experimental benchmarks was as follows:

1. The modified bin-stretching algorithm does produce the best overall performance of all the schedulers.
2. This advantage is most apparent for mixed message loads, which are typical of the load likely to be seen in an operational system.
3. However, the benefits from this scheduler do require a reasonably accurate estimate of the relative speed of the available machines in a heterogeneous computing environment.

These results confirm the theoretical predictions described in this paper, but highlight the need for realistic models of the available platforms to support a rugged, load-balancing scheduling method.

7. Conclusion and outlook

In this paper we have presented an enhanced version of the Bin-Stretching algorithm as a distribution strategy of jobs among hosts. The algorithm satisfies the business-driven requirements of a distributed message converter system like meeting of deadlines, priority processing, low response times and high throughput. The modified Bin-Stretching respects different host speeds and gives good results independent from the number of machines.

The algorithm behavior has been analyzed in different scenarios corresponding to different message distributions. The simulation results shows that the modified Bin-Stretching strategy gives in general better results then the well know FCFS strategy.

The general behavior has been verified on the real converter system. Also in this case the modified Bin-Stretching algorithm produces the best overall performance of all tested schedulers. The real system tests show also necessity of realistic system models for the cost estimation on different machines.

From the analysis arises further question that need to be addressed in the future to improve the algorithm. As scheduling results depend on the processing cost estimation further work should be done in this area, e.g. by using more sophisticated models. Further work can also be done at the algorithm itself. The additional configuration steps could be omitted if the algorithm would determine the optimal threshold itself during runtime. This would also make the algorithm more insensitive to changes of the message distribution.

8. References

[1] S. Albers. Better bounds for online scheduling. In Proc. 29th Annual ACM Symposium on Theory of Computing, pages 130--139, 1997

[2] Y. Azar; Online Load Balancing; Lecture Notes Iin Computer Science Vol. 1442; Springer Verlag, Berlin; 1998

[3] Y. Azar, O. Regev; Online Bin-stretching; Proc. of 2nd. RANDOM 1998, p. 71-81.

[4] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New Algorithms for an Ancient Scheduling Problem. In Proceedings of 24th Annual ACM Symposium on Theory of Computing, 1992, pp. 51-58

[5] J.Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt, J. Weglarz; Scheduling Computer and Manufacturing Processes, Springer Verlag, Berlin, 1996

[6] P. Bratley, M. Florian and P. Robillard; Scheduling with earliest start and due date constraints; Nav. Res. Log. Quart., Vol 18, p. 511-519; Dec. 1971

[7] Peter Brucker ; Scheduling Algorithms; Springer Verlag, Berlin, 1998

[8] Michael L.Dertouzos; Aloysius Mok. Multiprocessor On-line Scheduling of Hard Real-Time Systems. IEEE Transactions on Software Engineering, 15(12), December 1989.

[9] G. Galambos and G. Woeginger. An on-line scheduling heuristic with better worst case ratio than Graham's list scheduling. SIAM Journal on Computing, 22:349--355, 1993.

[10] R. Graham; Bounds for certain multiprocessor anomalies; Bell System Technical Journal 45; 1966

[11] J. Hergersberg; Bargeldloser Zahlungsverkehr durch Datenaustausch; Deutscher Sparkassenverlag GmbH, Stuttgart; 1997

[12] ISO 9735-1, EDIFACT - Application level syntax Rules, ISO, 1997

[13] T. Risse, A. Wombacher and K. Aberer; Efficient Processing Of Voluminous EDI Documents; Proceedings of ECIS 2000; Vienna; 2000; p. 343-350

[14] U. Schwiegelshohn, R. Yahyapour; Improving First-Come-First-Serve Job Cheduling by Gang SCheduling; Lecture Notes in Computer Science 1459, p. 180-198, Springer-Verlag, Berlin; 1998

[15] Jiri Sgall; On-line scheduling - A Survey; Online Algorithms: The State of the Art; Lecture Notes in Computer Science 1442, p. 196-231, Springer-Verlag, Berlin; 1998

[16] Anrew S. Tanenbaum; Modern Operating Systems; Prentice-Hall, London; 1992

[17] Chengzhong Xu; Francis C.M. Lau; Load Balancing in Parallel Computers; Kluwer Academic Publishing, Boston; 1997