Efficient Processing Of Voluminous EDI Documents

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

ABSTRACT

Business data is frequently exchanged between heterogeneous information systems using standard EDI formats like EDIFACT, X12 and in the future also XML/EDI. For inhouse use, data represented in these formats must be converted to inhouse data formats by message converters. With the growing usage of EDI, high volumes of data, like financial transactions, must be converted and processed efficiently and reliably.

Considering the complex hierarchical structure of EDI messages, message conversion is a complex data transformation problem. For efficient processing, the different conversion steps have to be executed in an optimized manner exploiting the available, typically distributed, processing architecture. In addition, the processing has to take into account different optimization goals, like maximizing throughput, minimizing delays and keeping deadlines.

We approach message conversion as a data management problem. First we show that the nested relational data model is adequate for representing the message structure and the transformation operations. Based on this we develop a cost model. This serves as input to an optimization strategy for the conversion processing that uses efficient scheduling strategies.

We present a novel scheduling strategy and give simulation results that show that they outperform alternative strategies for typical workloads of EDI converters.

This work is performed within the ESPRIT project POEM (No: 26.356). It is funded by the commission of the European communities. The Partners in POEM are GMD - German National Research Center for Information Technology (Germany), MOSAIC SOFTWARE AG (Germany), IT Innovation Center (Great Britain), WestLB (Germany), ABN Amro Bank (Netherlands).

I. INTRODUCTION

In times of growing networking of enterprises, the electronic implementation of business processes gains great importance. 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. Widely used formats are EDIFACT, X12 and in the future also XML/EDI. In addition national branch-specific formats, like DTA in Germany, play an important role. The enterprise side uses different inhouse formats. These are usually proprietary formats, which are developed and extended over the years. So the incoming and outgoing messages must be converted from the incoming format to the inhouse format, as well as from the inhouse format to the outgoing format. The volume of data each enterprise delivers and receives will grow rapidly in the next years. This fact together with additional business requirements (e.g. security services), forces the development of innovative new converter technologies to fulfill these requirements.

Within the POEM (Parallel Processing Of Voluminous EDIFACT Documents) project, a new concept for the conversion of financial EDIFACT message to the inhouse format and vice versa will be developed. Because of legal regulations it is expected that the number of EDIFACT messages and their size (from a few kilobytes to several megabytes) will grow. Thus processing these messages within a short time period, while observing reliability, scalability, and adherence to the business rules is of crucial importance. One of the most important business rules in the financial sector is the meeting of deadlines. This is important for the banks since they guarantee their customers that all incoming messages, which arrive within a given arrival time, will be processed till the next deadline.

The following sections describe the first results of analyzing the problem of EDIFACT message conversion and the architecture of a future parallel conversion system. In Section II an abstract message model capturing the common characteristics of the different data formats will be introduced. Section III describes the system architecture and our approach to the scheduling problems within the

conversion process. The paper ends with an outlook on future steps.

II. THE MESSAGE MODEL AND PROCESSING STEPS

We consider message conversion as a new type of data management problem. The approach we take in POEM is to use established notions and approaches of data management as far as possible and then extend them where needed to address the specific characteristics of the problem. This applies to the modeling of the message data, to the description of the message conversion process, and to the optimization of the message conversion process.

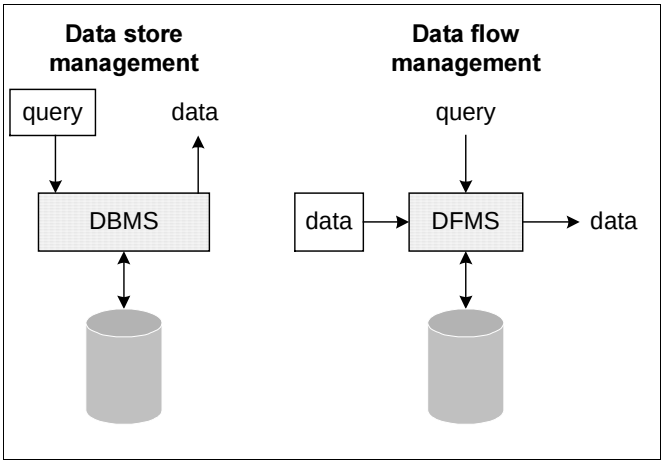

Fig. 1: Data store vs. Data flow management

The main difference to classical database management is that we deal with a data flow management problem rather than a data store management problem. This reflects the business dynamics of the application. While in a data store the process is driven by incoming queries, in a data flow the process is driven by incoming data while the queries are mostly static (see Fig. 1).

Our development of an advanced EDI message conversion system started with an analysis of the message formats used as incoming data and the processing steps during the conversion. The message formats used in the financial sector are:

- EDIFACT: ISO9735, Electronic Data Interchange for Administration, Transport and Commerce International [9]
- DTA: German proprietary format for financial data exchange [7]
- SWIFT: International standard for financial data exchange [13]
- IDOC: SAP's format for data exchange

The inhouse formats of the known users are mainly extensions of the DTA or SWIFT format. So the development can concentrate on these four formats.

Messages according to these formats have an implicit hierarchical tree structure. The tree structure is typically used to cluster single transactions within the message according to different criteria, e.g. the sender or the receiver of a financial transaction. Some have a very deep structure (e.g. EDIFACT) and others have a nearly flat structure (e.g. SWIFT). It turns out that a common abstract message model can be defined based on the *nested relational data model* [6][11]. It is sufficiently expressive to represent all of the occurring message structures and it contains the operations required for describing the types of conversions required in message converters. An example of the structure of a header of a message is given in Fig. 2. It consists of sets, which possess either one attribute or a tuple of attributes. The operations required to describe the transformation process are then the usual nested relational operations of selection, projection, join, nest, unnest and map together with sorting and aggregation.

Based on this message model the conversion process can be separated into the steps shown in Fig. 3. The incoming message is first analyzed to determine the syntax format (e.g. EDIFACT, IDOC) and size. After that, the message is parsed, which results in a semantic representation of the message as a nested relationally structured object. The unpacking step splits the hierarchical structure of this object into several independent objects of a neutral message model without nesting structure. Then, the aggregation step clusters the independent objects according to the requirements of the destination model. This representation can be sorted according to different business rules, e.g. sender, receiver and date. Finally, the instantiation step creates the syntactical representation of the semantic model and the resulting file can be generated and delivered.

The operations of this abstract processing model have to be executed in an optimized way with regard to the optimization goals of maximizing throughput, minimizing delays and keeping deadlines. To achieve the goals, some knowledge about the estimated processing costs is required. The abstract processing model is the basis to determine such a cost model for the operations (by measuring system performance, s.a. Section III.C.4). The cost model is used to estimate the costs for individual processing steps, which is the basis of the decision for the optimal execution strategy.

The existing system architecture and processing steps allow to optimize the processing by dynamic assignment of different processing steps to physical resources in a distributed system architecture. Thus the optimization strategy will rely, in the first place, on effective scheduling methods which in particular have to take account of the internal data structure of the messages.

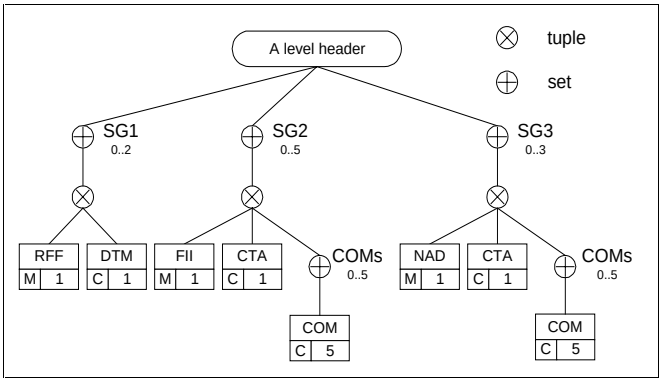

Fig. 2: Header structure of an EDIFACT message

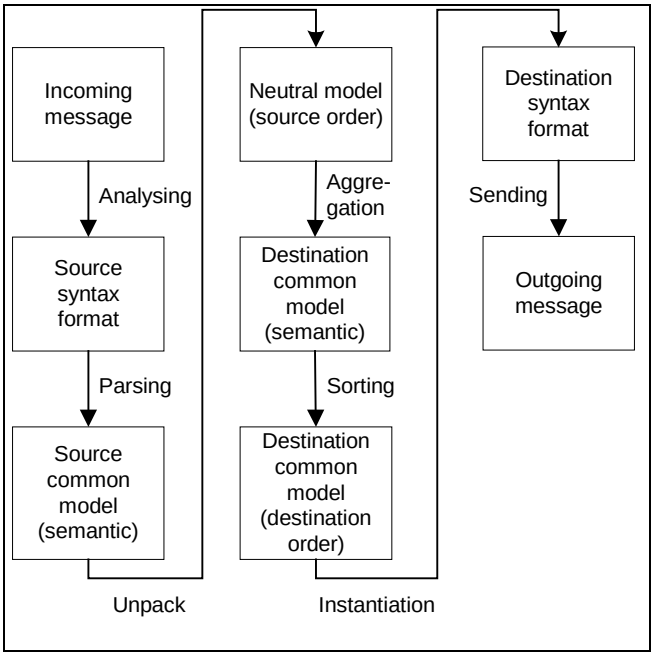

Fig. 3: Processing model

III. ARCHITECTURE AND SCHEDULING

The goal of the system architecture is to achieve the previously mentioned goals of reliability, scalability and high throughput. High throughput can be obtained by using multiple processors. This requires most of the conversion process to be done in parallel. Scalability means that a user can start out with the installation of a small machine and add processors or other machines to the system, later. The possibility to add different machines also increases the reliability of system. This is very important for banks, which guarantee their customers the availability of the system and certain processing times.

The range of possible system configurations is large. It begins with one-processor machines and ends with clusters of different machines with different performance characteristics. Many combinations are possible. This requires a flexible and effective scheduling mechanisms for distributing the incoming jobs on the available computing resources.

A. SCHEDULING LEVELS

The job of a message conversion must be scheduled to the available machines. The scheduler has to consider 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 small 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.

Dependent on the processing strategy, the last two requirements are in conflict. One strategy is to process as many files in parallel as possible. So each file is processed on one processor. This causes much higher delay for each file. In contrast, processing one file on as many processors as possible improves the response time for each file.

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. On this level the incoming jobs are distributed with respect to previously described goals to the available machines. In our processing model a job is defined as the conversion of a complete messages shown in Fig. 3. In our system model a job can only be assigned to one machine. This means that all information required for the processing is locally available on the machine. Moving a job to another machine is not allowed under normal circumstances because moving locally stored information and processing states are prohibitively expensive. The only exception of that is a machine failure.

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 in our system model is the arrival of an EDIFACT message with priority. In this special case the processing must start with as little delay as possible. So the processing of a running job must be suspended and later be restarted.

Each job belongs to a complete or only a part of a message. So it is possible to distribute the conversion of a message on several hosts. This is useful for very large messages because more processing capacity is available for the conversion process. For small messages it makes less sense because the administrative overhead grows in relation to the processing time. Our approach is to decide this dynamically.

The second level of scheduling is the *local scheduling* which is performed on each machine separately. At this level the jobs assigned by the global scheduler are split into the individual tasks. A task represents one step of the processing chain. Some restrictions applying to a job are also valid for a task. So a task cannot be moved to another machine. But the preemption of tasks is allowed. Depending on the estimated processing time (Subsection C.4) and the different possibilities for parallelization for each task the number of required processors can be determined. After that the execution of the tasks begins. At this point the local scheduler is tightly coupled with the operating system (OS) scheduler [12]. The OS scheduler distributes the tasks to the individual processors, while the local scheduler controls the execution order.

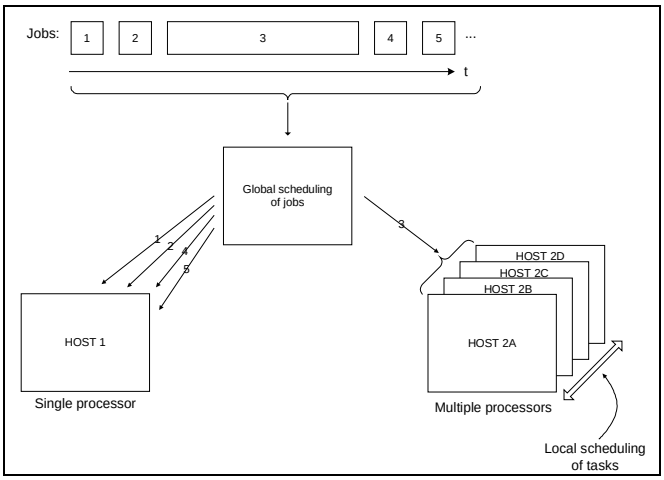

Fig. 4: System architecture and scheduling levels

B. *CLASSIFICATION OF SCHEDULING PROBLEMS*

In the literature there exist many algorithms for solving scheduling problems. They are mostly classified according to the notation system described in [2] [3]. By this notation, the machine environment, the characteristics and restrictions of the tasks and the objective function (optimality criteria) can be described in a three term expression $\alpha \mid \beta \mid \gamma$ in which the first two terms can be further subdivided and such that the coding scheme has the form $\alpha_1 ... \alpha_n \mid \beta_1 ... \beta_m \mid \gamma$. In most cases some of the variables are empty ($\varnothing$) and do not appear.

This classification is used for static scheduling problems. This means that all parameters to the scheduling problem are known at start time of the scheduling algorithm. In the POEM scenario the problems (messages) arrive at a unknown future time. So scheduling has to be done with the available partial knowledge in runtime (Online scheduling). The schedules have to be updated every time a new message arrives. To describe this we extend the classification by an additional parameter, the 'arrival time'. This extension is based on the definition of a dynamic job shop scheduling in [15].

Arrival times

- $\varnothing$ static: all jobs are ready at start time 0
- D deterministic dynamic: the jobs arrive at known future time
- S stochastic dynamic: the jobs arrive at unknown future time

With this definition the problems can be classified in the following way:

Global scheduling

$$R, MPM \mid pmtn, d_j, S \mid L_{max}$$

- R - The resources are unrelated multi-purpose machines.
- MPM - The jobs can be executed on all available machines, but can only be assigned to exactly one machine.
- pmtn - Preemtion for the scheduling is allowed.
- d_j - Each job has a deadline.
- $\varnothing$ - The jobs are independent.
- $\varnothing$ - The jobs have arbitrary processing times.
- S - Stochastic dynamic arrivals of jobs.
- L_{max} - Minimizing the lateness is the goal of the optimization.

Local scheduling

$$P \mid pmtn, d_j, chains, S \mid L_{max}$$

They are the short forms for:

- P - The resources are identical parallel processors.
- pmtn - Preemption for the scheduling is allowed.
- d_j - Each job has a deadline.
- chains - The precedence constrains for the tasks forming a chain.
- $\varnothing$ - The tasks have arbitrary processing times.
- S - Stochastic dynamic arrival of jobs.
- L_{max} - Minimizing the lateness is the goal of the optimization.

C. *ANALYSIS OF SCHEDULING ALGORITHMS*

The arrival-time parameter makes the scheduling problem a dynamic one, but most available algorithms are used to solve static scheduling problems. Beside load-balancing strategies, dynamic scheduling problems are little investigated compared to static scheduling problems.

Load-balancing strategies have the goal to ensure equally loaded systems [16]. Because they mainly decide on the load of the system (where to assign a job), they do not take account of other optimization goals like e.g. deadlines. The

presented scheduling problems require some knowledge about the waiting jobs and their influence on the execution order. In general these goals cannot be achieved by load-balancing.

Hence a dynamic scheduler is required to achieve these goals. A first approach is to divide the dynamic problem into several static problems [14]. In this way static scheduling algorithms can be applied to solve the dynamic scheduling problem. Every time a new job arrives, a new schedule will be calculated. All pending jobs at this moment are used for the specification of the scheduling problem. The result is a schedule, which is valid till the next job arrives. Then it must be recalculated or updated. Because the schedules have to be calculated very often, the computational complexity has to be very low.

A second approach is to use online scheduling algorithms. The algorithms schedule incoming jobs without a complete recalculation. In this case, they are similar to load-balancing algorithms but they use further knowledge about the pending jobs to achieve different goals.

C.1. Simulation of scheduling algorithms

The behavior of scheduling algorithms is well known for static problems, but when applied in dynamic situations, as previously described, their behavior is not understood as well. Thus we started to evaluate them using a scheduling simulator. The scheduling simulator can simulate different scheduling algorithms in different host configurations in a way they are expected in the future POEM system. It can use different strategies for generating jobs, e.g. random or realistic. The generated jobs can also be stored for repeating tests with the same data.

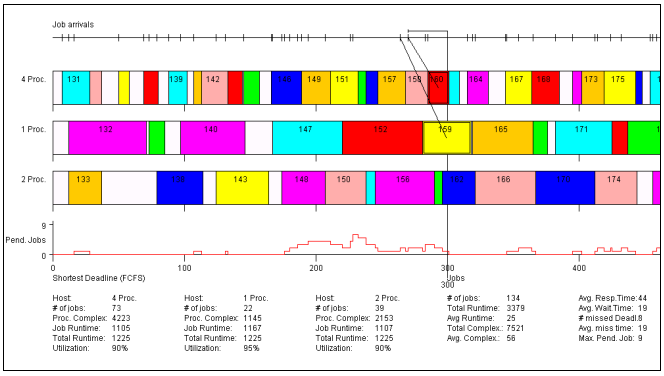

Fig. 5: Example schedule (Shortest Deadline First)

Fig. 5 shows an example schedule generated by the simulation of the global scheduler. It contains a Gantt chart for each machine, a time line with marks of the arrival time of each job, a graph that shows the number of pending jobs at each time and some statistics. The statistics contains information about, e.g. the host utilization, total runtime, and average runtime. The simulator can also collect statistics of several simulation runs to get a general overview of the scheduler.

The simulator is written in Java. Its object-oriented design allows to easily integrate different scheduling algorithms or to add new job generators.

C.2. Analysis of the global scheduling algorithms

Though the global scheduling problem has been classified as being preemptive, we treat it first as a non-preemptive problem, since this reflects the normal mode of operation.

Non-preemptive scheduling problems are known to be NP-hard [2] [3]. The following (heuristic) algorithms have been analyzed with regard to the objectives L_{max} (minimizing the lateness) or C_{max} (minimize runtime).

First Come First Serve (FCFS)

This strategy is the simplest form of list scheduling algorithms [5]. The jobs are executed in the order they arrive at the system. If a host is idle, it gets the next job from the waiting queue. The strategy does not consider different host speeds or the deadlines of the jobs.

If the jobs arrive in the order of their deadlines the strategy is similar to the Shortest Deadline (SDF).

Longest Processing Time First (LPT)

This algorithm is another type of list scheduling. It is an approximative solution for $P \mid\mid C_{max}$ (identical hosts, minimize runtime). The jobs are sorted by their processing time in a non-increasing order. In every step the first available processor gets the next job of the list. The main disadvantage of this strategy is that large jobs are preferred. So small jobs could be heavily delayed.

Shortest Deadline First (SDF)

This strategy is similar to the previous one. But it is an approximation for $P \mid\mid L_{max}$ (identical hosts, minimize lateness). The algorithm sorts the jobs by their deadlines and arrival times. As expected, this is a good strategy for minimizing the lateness. On hosts with different performance it often happens that large jobs are processed on the slower machine and vice versa.

Bin-Stretching

The bin-stretching approach belongs to the group of online algorithms. The original algorithm [1] first categorizes the hosts into groups of underloaded, normally loaded and overloaded systems based on the future load of the host. After that, the least loaded system will be selected. The disadvantages of the original version are that the algorithm is restricted to identical hosts, that it does not consider other goals. Nevertheless, we have modified the strategy to avoid these problems. First results are presented in Section IV.

The analyzed strategies achieve the goals of the global scheduler only partially. So in Section IV we present a combination of BinStretching and list scheduling, which achieves the goals of the global scheduling.

C.3. Analysis of the local scheduling

The local scheduling can have a preemptive solution. So a good approximative algorithm can be found more easily. The problem of the dynamicity remains the same as in the global scheduling problem. Another important characteristic of the problem is that the tasks are not independent. This means that one task requires some or all results of the previous processing step. The processing steps are organized as a chain as shown in Fig. 3. Known algorithms for this type of scheduling problems are described below. The algorithms are too complex to describe them here briefly, so only the problems they solve are mentioned.

Level algorithm for preemptive scheduling [8]

This strategy approximates Q | chains; pmtn | C_{max} (uniform hosts, precedence relation between tasks forming a chain, preemption is allowed, minimize runtime).

Priority scheduling [10]

This is an approximation for P | intree; pmtn | L_{max} (identical hosts, precedence relation between tasks forming a tree which is directed to the root, preemption is allowed, minimization of lateness).

The 'Critical Weight' Algorithm [4]

The algorithm solves a problem of the type 'P | tree; pmtn | C_{max}' (identical hosts, precedence relation between tasks forming a tree, preemption is allowed, minimize runtime).

The behavior of these algorithms in dynamic situations is currently not simulated so, actually, no results are available. But it is expected that the results are similar to the analysis of the global scheduler.

C.4. Estimation of processing times

All algorithms described need to estimate the processing times of the job or task. Such an estimation is not easily given since it depends on a lot of factors, e.g. other activities on the machines. Currently, we develop such a cost model starting from the abstract processing model of a generic message converter, identifying the required processing steps (Section II). For each of these steps, we determine the complexity of the algorithms. We also gather statistical information of processing in real situations. From this we develop an estimation function which calculates the expected processing time in relation to the incoming job size. The variance in the real processing times are described by a standard deviation. From this the scheduler can also derive a standard deviation of the expected processing times.

IV. General scheduling approach

Our general scheduling approach first splits the problem into normal processing and priority processing. As mentioned in the previous section, messages with priority have to be started within a small time range. They must be assigned immediately to an available host, or a job on a fast host must be preempted. Hence, a FCFS strategy with the possibility to preempt running jobs solves this problem.

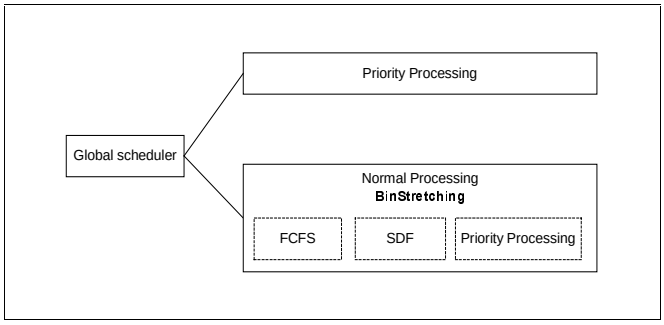

Fig. 6: General scheduling approach

For the distribution of normal jobs, a combination of two strategies is used. The first step is to select an appropriate host for the job by using a modified version of the BinStretching algorithm [1]. The original BinStretching was extended in a way that it takes into account different host speeds. A brief description of the algorithm can be found in the text box below. The second step is to order the jobs according to other goals, e.g. deadlines, priorities, etc. This step is similar to list scheduling [5] on a single host.

Modified BinStretching Algorithm

First we need some definitions. A host is said to be short if its current load is at most $\alpha \cdot L_{max}$ (L_{max}=maximum load of all hosts). Otherwise, it is tall. The value of α can be in the range between 0 and 1. It describes the threshold between a short and tall loaded host. The value of α influences the quality of the resulting schedule.

After the arrival of a new job the following disjoint sets are defined. The processing times are always scaled according to the host speed.

- S_1 contains all hosts whose total processing times are short or remain short if the current job is placed on them.
- S_2 is a set of hosts that are short but become tall if the job is placed on them.
- S_3 contains all remaining jobs

In the next step of the algorithm a host is selected out of these three sets.

- Put the job on the fastest non-empty host from the set

S_1. If S_1 contains only empty hosts then put the job on the fastest empty host.

- If $S_1 = \varnothing$ then put the job on the least loaded host after assignment from set S_2.
- If $S_1 = S_2 = \varnothing$ then put the job on the least loaded host after assignment from set S_3.

After the assignment of the job to the host, the waiting queue has to be resorted according to the additional goals taken into account.

D. *Simulation Results*

To analyze the behavior of the modified BinStretching algorithm, we performed several simulations with the scheduling simulator introduced in Section C.1. We can present here the results for one of the simulations with one host combination and job distribution.

The simulated system consists of a six and two processor host. This means that the first host is three times faster than the second one. The incoming job sizes are normally distributed with a very low average size. The biggest job requires 20 times more effort to process than an average job.

The charts in Fig. 7 and 8 show histograms of the differences in response time between FCFS and the modified BinStretching. Positive values indicate an advantage for the BinStretching strategy, while negative values indicate an advantage for FCFS. The charts also contain a comparison of different α values.

With BinStretching, most of the jobs have the same or shorter response time. This can be seen in Fig. 7. Interesting and important for the converter system is that large jobs gain a lot from the algorithm. A histogram for large jobs is shown in Fig. 8. It shows that many of the large jobs are processed much quicker with modified BinStretching. We also analyzed how jobs keep their deadlines and found similarly positive results for modified BinStretching.

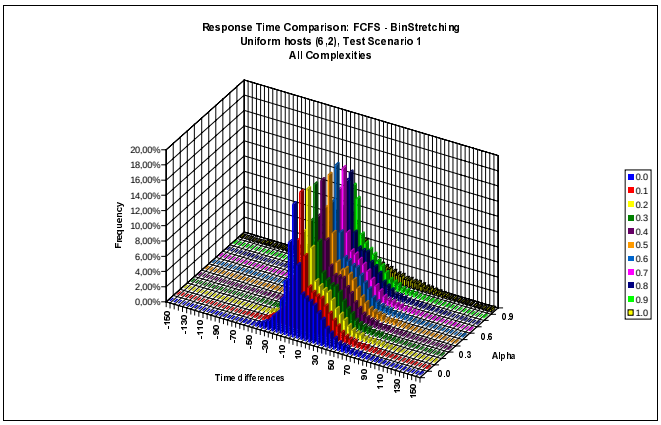

Fig. 7: Response time comparison for all jobs

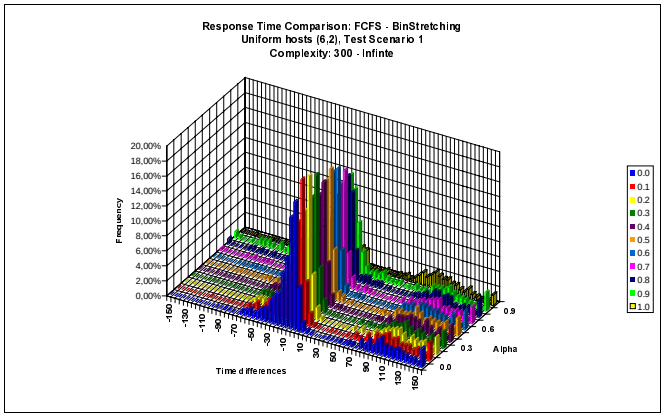

Fig. 8: Response time comparison for large jobs

When analyzing how jobs are distributed to hosts it shows that 50% of the small jobs are assigned to the slow host in this experiment. But over 90% of the large jobs are processed in the fast machine. Hence, the algorithm takes into account the host speeds and works as expected.

With respect to the choice of a value of α, values between 0.1 and 0.4 seem to give the best results for the simulated job distribution and host combination.

V. Conclusion and Outlook

This paper describe the first steps and results of the development of a new systematic approach for converting messages from one format to another. This requires a formal message model in which the abstract design of a generic message converter can be described. It is used for the identification of the required processing steps. After that, the system architecture is defined based on the goals of reliability, scalability and high throughput. The architecture requires a powerful scheduling component, which is one of the core problems of the approach. Two types of scheduling levels are identified, global and local scheduling. The dynamic character of the scheduling problems and the complex estimation of processing times make it difficult to develop an optimal solution. The presented modified BinStretching approach gives good results with respect to all identified goals. The algorithm has to be further analyzed regarding other host combinations and job distributions. Another problem is the variance of estimated processing time. The scheduling approach has to be tested how it reacts on longer or shorter real processing times.

References

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

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

[3] Peter Brucker ; Scheduling Algorithms; Springer-Verlag, Berlin-Heidelberg-New York; 1998

[4] Teofilo F. Gonzales and Donald B. Johnson; A new algorithm for preemptive scheduling of trees; Journal of ACM, Vol. 27, No. 2, p. 287-312; 1980

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

[6] Güting, R.H.; Zicari, R.: An Introduction to the Nested Sequences of Tuples Data Model and Algebra., in: Abiteboul, S.; Schek, H.-J.: Nested Relations and Complex Objects in Databases, Springer-Verlag, Berlin-Heidelberg-New York; 1987

[7] Hergersberg, Johannes: Bargeldloser Zahlungsverkehr durch Datenaustausch, Deutscher Sparkassenverlag GmbH, Stuttgart; 1997

[8] Horvath, Edward C., Shui Lam, Ravi Sethi; Level algorithm for preemptive scheduling; Journal of ACM 24; ACM; 1977, p. 32-43

[9] ISO9735, Electronic Data Interchange for Administration, Transport and Commerce International; ISO; 1988

[10] E. L. Lawler; Preemptive scheduling of precedence-constrained jobs on parallel machines; Deterministic and Stochastic scheduling; D. Reidel Publishing; 1982, p. 101-123

[11] Levene, M.; The Nested Universal Relational Database Model, Springer Verlag, Berlin-Heidelberg-New York; 1991.

[12] Rajkumar Buyya; High Performance Cluster Computing Vol.1; Prentice Hall, New Jersey; 1999

[13] S.W.I.F.T. User Handbook, S.W.I.F.T., La Hulpe; 1997

[14] Jiri Sgall; On-line scheduling - A Survey; Online Algorithms: The State of the Art, eds. A. Fiat and G. J. Woeginger, Lecture Notes in Computer Science 1442, p. 196-231, Springer-Verlag, Berlin-Heidelberg-New York ; 1998

[15] Shyh-Chang Lin, Erik D. Goodman, William F. Punch; A Genetic Algorithm Approach to Dynamic Job Shop Scheduling Problems; Proceedings of the 7th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Francisco; 1997

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