COST-BASED BUSINESS PROCESS DEPLOYMENT ADVISOR

Steffen Preissler, Dirk Habich and Wolfgang Lehner
Database Technology Group, Dresden University of Technology, Dresden, Germany

Keywords: SOA, Process, XML, Deployment.

Abstract: Service-oriented environments increasingly become the central backbone of a company's business processes. Thereby, services and business process flows are loosely coupled components that are distributed over many server nodes and communicate via messages. Unfortunately, communication between SOA components using the XML format is still the most resource- and time-consuming activity within a traditional business process flow. This paper describes an approach that considers these transfer-costs and advises a process and service placement for a given set of server nodes to partially eliminate remote message transfers. Furthermore, experiments have been conducted to show its feasibility.

1 INTRODUCTION

Nowadays, service-oriented architectures have been adopted as architecture paradigm in company-wide networks. Following this paradigm, reusable pieces of code are exposed as services over a variety of server nodes. They communicate via XML messages and by the use of workflow languages like WSBPEL (OASIS, 2007), these services can be orchestrated to more comprehensive processes.

A typical network topology setting for current SOA implementations is shown in Figure 1. It comprises the following two characteristics: *First*, server nodes ($n1 \ldots n4$) host services ($s1 \ldots s4$) and processes ($p1 \ldots p3$). These server nodes are interconnected with and registered at an enterprise service bus (ESB). With that registration the physical location of such components and its assignment to respective server nodes becomes transparent to clients. *Second*, the execution of processes is realized by process engines that usually reside on dedicated server nodes. Although prototypes for decentralized process execution exist (e.g., (Chafle et al., 2004)), as of today, the majority of process engines execute their process instances centrally (see processes $p1$, $p2$ on node $n3$) (Martin et al., 2008). Each process typically interacts with a set of services to access or modify the data they provide. For our example in Figure 1, the set of particular services s_i for a process p_i is: $p1 = \{s1, s2, s3\}$, $p2 = \{s2, s4\}$ and $p3 = \{s2, s5\}$ which is also depicted by the colored dotted lines.

One challenging bottleneck is the resource-intensive XML message exchange due to XML conversion oerhead and the increased data volume XML text infers (Habich et al., 2007). Although approaches exist that deal with both problems (e.g. (Suzumura et al., 2005) and (Ng, 2006)) they only reduce transfer costs by reducing the "'thickness"' of the dotted lines in Figure 1.

A better solution would be to eliminate complete service calls by nearby placement of processes and services. Therefore, this paper describes the idea of a business process deployment advisor that advises optimized process placements to server nodes regarding their corresponding services and their different communication costs.

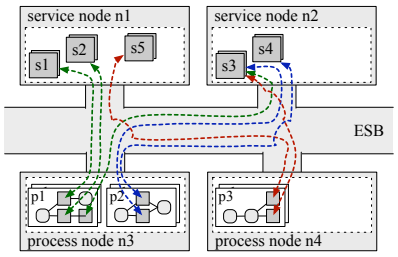


Figure 1: SOA with centralized process execution.

2 DEPLOYMENT ADVISOR OVERVIEW

In this section, we will give a brief overview of our *process deployment advisor* approach. The core as-

Preissler S., Habich D. and Lehner W..
COST-BASED BUSINESS PROCESS DEPLOYMENT ADVISOR.
DOI: 10.5220/0003490602110216
In *Proceedings of the 13th International Conference on Enterprise Information Systems* (ICEIS-2011), pages 211-216
ISBN: 978-989-8425-53-9

sumption thereby is that today's server nodes hosting services are not fully utilized in terms of CPU cores and main memory. The concurrent execution of services and processes on one server seems reasonable and will reduce remaining network transfer. Additionally, these servers may run the same execution environments for both services and processes (e.g. application server like JBoss (JBoss, 2010)). Thus the communication between process and service can be based on shared objects in main memory rather that XML text messages. This will eliminate XML conversion and significantly reduce resource utilization.

Therefore, the main goal of our advisor is to place processes near to dependent services based on there communication costs. This 1) reduces network transfer times and latency for relocated processes, 2) eliminates resource-intensive XML conversion and latency significantly if process and service are executed in the same software environment and 3) frees network bandwidth on the ESB for other tasks.

2.1 Advisor Meta Model

Our advisor meta model describes the entities and their interaction that are the base for our advisor algorithm in Section 3: a set of processes P, a set of services S, a set of service bundles B and a set of server nodes N.

The set of processes P represents the business processes we want to reassign to server nodes. Thereby a single process plan p is defined as $p = (A, S_p)$, where A is the set of activities with $A = (a_1, \ldots, a_i, \ldots, a_k)$ that are connected as control flow in an acyclic, directed graph and S_p with $S_p = (s_1, \ldots, s_i, \ldots, s_l)$ is the set of services that P interacts with. Note that S_p is actually a subset of S. For A, we distinguish two classes of activities in p_i for cost purposes based on (Böhm et al., 2009): *interaction-oriented* and *non-interaction-oriented* activities. *Interaction-oriented* activities comprise `receive`, `invoke` and `reply`, whereas *non-interaction-oriented* activities comprise e.g., `assign`, `switch` or `fork`. Processes interact with services using their *interaction-oriented* activity `invoke`. In Figure 2, process plan $p1$ interacts with services $\{s1, s2, s3\}$ and $p2$ with $\{s2, s4\}$.

A services s in S is defined as $s_i = (d_{in}, d_{out}, f)$. Thereby, d_{in} describes the input data and d_{out} is the output data with $d_{out} = f(d_{in})$. Note, that d_{in} an d_{out} have direct effect on communication costs in processes and are therefore used to calculate communication costs. Furthermore, services are considered as black box only exposing data structures and coarse-grained execution statistics.

Since most services and their locations have evolved historically, there may be dependencies between services. As an abstraction layer between services and their actual execution node we introduce *service bundles*, where services in one bundle are logically grouped and may, if relocated, be relocated as one atomic block. Thereby a service bundle b can comprise all services on one node or just a subset of it with $b = (s1, \ldots, sk)$. Furthermore, it is assigned to a specific server node and can be marked as moveable or not.

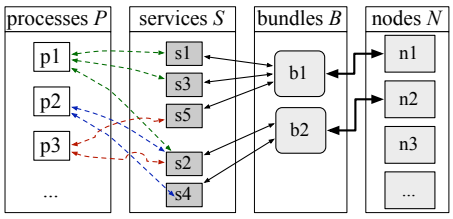


Figure 2: Process Deployment Advisor Meta Model.

Finally, servers in N are the physical execution environment for processes and services. They are considered heterogenous in terms of CPU and main menory but equal in their type of network interconnect.

2.2 Statistics and Cost Model

To be able to evaluate every meta model entity in terms of resource consumption and communication costs, we have to gather statistics and define a cost model. For the statistics, on every server node, the node itself and its components (services S and processes P) are monitored and all statistics are sent to the advisor component and stored for further use. If no statistics for a specific node-component pair have been recorded, the statistics for central execution is used.

Execution Statistics. For *services*, we monitor basic statistics for every service s_i like workload $M(s_i)$ and average execution time $T_{exec}(s_i)$. For *processes*, a more fine-grained monitoring is required. Beside the actual workload $M(p_i)$ we monitor the average execution time $T_{exec}(a_i)$ for all k activities in process plan p_i. Furthermore, we subtract the waiting time $T_{wait}(a_i)$ from $T_{exec}(a_i)$ for *interaction-oriented* activities to get their plain processing time. Additionally, for *interaction-oriented activities* in p_i, average input and output cardinalities $|d_{in}|$ and $|d_{out}|$ are monitored. This includes the number and position of XML elements to cover workload-dependent repetition groups in arrays. Although cardinalities might be useful for all activities and for general process plan

optimization, we only consider it for our *interaction-oriented* activities, since it directly affects communication costs.

Communication Costs. Using the data from cardinality monitoring and the specified message schemata in an *interaction-oriented* activity a_i, its communication cost $C_{com}(a_i)$ can be analyzed with different metrics. For example, data size-related metrics like number of bytes or number of nodes in a document can be used solely. Such metrics are reasonable if comparing pure network transfer between activities or processes. Nevertheless, the core idea behind our XML cost model is to compare the message transfers of processes in terms of CPU utilization and a potential benefit in a local, *object-based* service calls. Thus, a more sophisticated cost computation is needed instead of using pure data size-related metrics. Experiments have shown that the processing time of an XML conversion depends on its structural characteristics. Hence, our cost model considers the number n_e of element nodes e , the number of attributes n_a, the number of hierarchy levels l with $l \geq 1$ and the distribution of n_e element nodes over hierarchy levels l. Communication costs are computed as follows:

$$C_{com}(a_i) = |d_{in}| + |d_{out}| \quad \text{with}$$
$$|d_{in}| = \lambda_{e|Ser} \cdot n_e \cdot n_a$$
$$|d_{out}| = \lambda_{e|Deser} \cdot (n_e - (l-1)) + (l-1)^{\lambda_l} + \lambda_a \cdot n_a$$

Thereby $\lambda_{n|Ser}$ and $\lambda_{n|Deser}$ denotes hardware-specific constants for processing an element node for serialization and for deserialization respectively. Furthermore, λ_l and λ_a are hardware-specific constants for deserializing these nodes accordingly. For a starting point, the constants are derived from our server node for central process execution. Note that the presented equations only consider process side costs for the invoke activity. Thus, the input dataset d_{in} *to* the service is serialized with linear costs whereas the output dataset d_{out} *from* the service has to be deserialized on process side considering structural aspects.

2.3 Problem Definition

Having described the prerequisites of our deployment advisor, we now formally defined the process placement problem that we want to solve:

Definition 1. Process Placement Problem. *Assume a set of process plans P with $P = (p_1, \ldots, p_k)$, a set of server nodes N with $N = (n_1, \ldots, n_l)$ and a set of services S with $S = (s_1, \ldots, s_m)$ where every service s_i is associated with a server node n_j. Furthermore, let $M(c_i)$ describe the workload of a component (process or service) c_i as the number of input messages within a predefined time period Δt that all execute instances of c_i. Finally, let $R_N(n_i)$, $R_P(p_i)$ and $R_S(s_i)$ denote the maximum resource capacity of node n_i and the average resource consumption of process plan p_i and services s_i in the same time period Δt respectively. Then, the process placement problem describes the search for a process deployment mapping of processes to server nodes that reduces or eliminates the maximum possible network communication in comparison to central (full remote) execution, while not exceeding the servers resource capacities for a given workload.*

Although stated in this problem description, this paper does not cover details about workload and resource-aware process placement. Instead, it focuses on the basic models and the algorithm to minimize network transfer.

3 BASIC PROCESS PLACEMENT

In this section we will introduce the basic process placement algorithm. It computes the server node for a given process with the largest communication overhead (and thus the largest network transfer saving with local XML transfer or largest CPU-resource saving with local shared objects).

Based on the communication cost model C_{com} defined in the previous section, we now define our first optimization goal: *Minimize overall transfer costs between processes and services*. Thereby, we focus on the nearby placement of processes to services to eliminate these costs. In a first step we assume, that we are not able to relocate services and that they are *fixed* to their current server nodes. Thus, to reduce network transfer costs, we move processes from their central execution node to one of the respective service nodes that are included in some of a process's invocation. That is, we bundle this process and its execution with the set of services on that node. For a given process plan p, Algorithm 1 retrieves the list of server nodes in the order of decreasing communication cost and returns the topmost server in the list. In other words, the returned node produces the most communication costs and the process execution on that node is most reasonable.

In more detail, Algorithm 1 takes a process plan as input and retrieves all invoke activities and corresponding nodes (lines 3-8). In the second part, every involved node is analyzed (line 10). In line 12 - 16, all invoke activities a_j from p that call a service of the current node are considered and their costs $C_{com}(a_j)$ are added to the overall costs of node n_i (lines 15 and 17). If the execution environment of process and

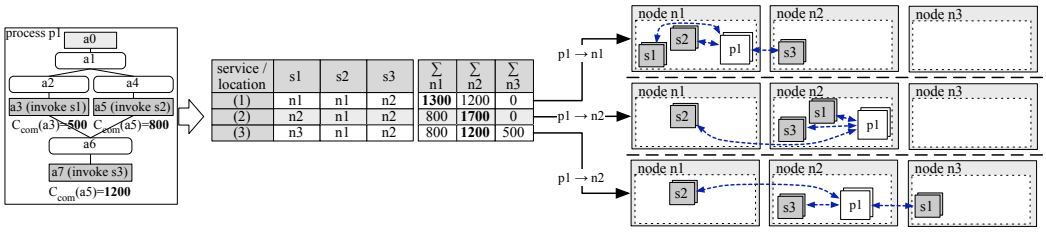


Figure 3: Static Process Placement Algorithm Example.

service are equal (line 14), shared object calls can be used. Thus, this communication (and this node) is rated more important with adding factor λex with $\lambda ex > 1$ to the costs (line 15). Thus, it is more likely that p is placed at n_i. An ordered list with descending process communication costs is created (line 18) and the node involving the most communication cost is returned as the node where the process should be placed (line 19).

Algorithm 1: RetrieveBestNodeForProcess.

```
Require: process plan p
 1: A_I ← ∅, N ← ∅
 2: // part 1: extract invoke activities
 3: for i = 1 to |p.A| do                    // ∀ activities in p
 4:   if a_i.type ==' invoke' then
 5:     A_I ← A_I ∪ a_i
 6:     N ← N ∪ node(a_i)
 7:   end if
 8: end for
 9: // part 2: compute cost for every node
10: for i = 1 to |N| do                      // ∀ nodes n_i
11:   n_i.cost := 0
12:   for j = 1 to |A_I| do                  // ∀ invoke activities
13:     if n_i.nid == node(a_j).nid then
14:       if p.envid == service(a_i).envid then
15:         n_i.cost := n_i.cost + C_com(a_j) · λex
16:       else
17:         n_i.cost := n_i.cost + C_com(a_j)
18:       end if
19:     end if
20:   end for
21: end for
22: nlist ← order(N, N.comcost, DESC)
23: return nlist[0]
```

An example is shown in Figure 3. Thereby, process $p1$ communicates with three services. It concurrently fetches data (activities $a3$ and $a5$, joins them ($a6$) and stores them remotely ($a7$). The denoted communication costs C_{com} have been computed for all three invoke activities: $a_3 = 500$ for retrieving the orders, $a_5 = 800$ for retrieving the invoices and $a_7 = 1200$ for storing the joined data in service $s3$. These activities are analyzed and the corresponding remote service nodes retrieved. Three possible service distributions of the services $s1$ to $s3$ are depicted in row (1), (2) and (3) in the center table. For example, in the first distribution, services $s1$ and $s2$ are located on node $n1$ whereas service $s3$ is located on node $n2$. For every single service distribution, the sum of communication costs associated with the server node differs. In the end, the algorithm chooses the node with the maximum communication cost for the specific process. This is done for every existing process plan.

The execution of this basic algorithm computes the best node for a specific process. Although this eliminates the maximum possible network transfer, frees network capacity and improves latency for remaining tasks on the ESB, this setting does not consider process workload and resource constraints that may increase process latency. A workload-aware extension of this base algorithm is subject to future research.

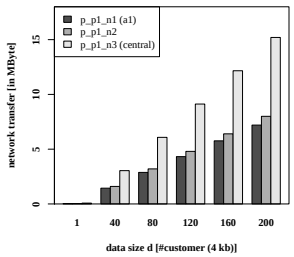
4 EVALUATION

In this section, we present selected results of our experimental evaluation. We implemented our advisor algorithm and the processes using Java 1.6 and the Web service framework Axis2[1]. The overall setting follows Figure 1 with all services running on nodes $n1$ and $n2$ respectively with $n1 = \{s1, s2, s5\}$ and $n2 = \{s3, s4\}$. Processes p1 and p2 run centrally on node $n3$ as starting point. All experiments were executed on synthetically XML data and were repeated 50 times for statistical correctness.

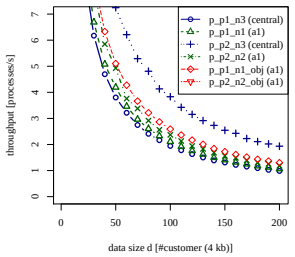
4.1 Performance Measurements

We present three experiments that show the effect of our communication-aware process placement. For general cost savings in terms of reduced network transfer, Figure 4(a) depicts the result for one single process execution of p_1 and a varying input data size d. Thereby, one customer information (3kb) is augmented with orders and invoices at service $s1$ and $s2$

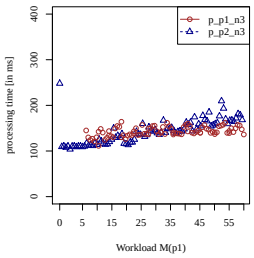
[1] http://ws.apache.org/axis2/



(a) Produced Network Transfer p_1.



(b) Single Throughput of p_1 and p_2.



(c) Performance Concurrent Process Execution.

Figure 4: Performance Measurements.

(approximately 17kb for each response, that is in average 10 invoices and 10 orders per customer response) and the joined result is sent to service $s3$ (40kb). As illustrated, the central execution on node $n3$ (see `p_p1_n3`) naturally exhibits the most communication costs with increasing data size to communicate. For example, if one process instance queries 40 customer ids, approximately 4MB are transferred over the network. For nearby placement of p_1 to services $s1$ and $s2$ on node $n1$ (see `p_p1_n1`) and to service $s3$ on node $n2$ (see `p_p1_n2`) as suggested by our base algorithm, significant network transfer reductions can be achieved. Remember, the depicted network volume is for one single process instance execution.

Before we consider different workloads for both process types, we evaluate the *single performance* of p_1 and p_2. We execute them on every node $n1$ to $n3$ separately and use local XML transfer. Figure 4(b) depicts their throughput per second with different input data sizes. What can be observed is that executing p_1 on node $n1$ (`p_p1_n1`) performs generally better than remote execution (on $n3$, `p_p1_n3`), e.g. 53 processes per second against 48 with $d = 40$ customer. This is due to eliminating network transfer to the respective services for the most costly communication activities in p_1. Since XML serialization and deserialization is the most resource consuming task and since it still takes place for local service calls, the benefit in throughput enhancement on node $n1$ is not significant. In contrast, throughput of process p_2 decreases on node $n2$ `p_p2_n2`) (where all respective services of p_2 reside) since $n2$ has lower resource capacities to execute process p_2 and its services s_3 and s_4 in conjunction. If the processes can use shared objects for local service calls, throughput increases significantly (see `p_p1_n1_obj` and `p_p2_n2_obj`). Nevertheless, Algorithm 1 advises the placement of p_1 to $n1$ and p_2 to $n2$ since it uses the communication cost model solely to determine placement.

To show the negative effect of concurrent process type executions we place $p1$ and $p2$ on node $n3$, fix data size d for both types to 20 customer and execute process p_2 with a constant workload of 20. For p_1, we increase the workload from 5 to up to 60 in fixed time intervals. Figure 4(c) shows the performance of both process types. What can be observed is that at a workload of 15, the execution of p_1 affects the execution of p_2 and both runtimes increase. Furthermore, the variance for these executions also increases due to heavy scheduling between these process types.

5 RELATED WORK

To best of our knowledge, we are not aware of any work concerning communication-aware placement strategies for services or processes in heterogeneous SOA environments. Of course, there exists a lot of work touching aspects of our approach. As for distributed workflow management, traditional document-based workflow systems like (Alonso et al., 1997) or (Bauer et al., 2003) already considered the nearby execution of tasks and data. While (Alonso et al., 1997) explicitly separated control and document flow to provide remote tasks with required document data, (Bauer et al., 2003) divided its execution network into domains and used hash-based functions to balance process fragments within the domains. Nevertheless, within domains, the fragments were placed randomly on server nodes and communication costs between nodes were not considered at all. In SOA environments, approaches based on process fragmentation have also been proposed. (Chafle et al., 2004) and (Khalaf and Leymann, 2006) distribute centralized process orchestrations using automatic generation of distributed process fragments. This is similar to query plan partitioning in DBMSs. Nevertheless, they also do not consider a cost model or placement strategies for their fragments. In the area of service communication many approaches reduce

XML conversion using differential encoding (Suzumura et al., 2005; Werner et al., 2004). Other approaches optimize the network transfer by the use of compression techniques, e.g.(Maneth et al., 2008) or more compact message formats (Ng, 2006). Although these approaches increase XML performance significantly, they only reduce, not eliminate, the overhead of this communication.

6 CONCLUSIONS

In this paper, we presented the first step for a *business process deployment advisor* that eliminates network communication and, if possible, XML conversion between processes and its services by shared objects. It advises the placement of given processes to dependent services and server nodes with the focus on communication reduction. In a first step (and as the focus of this paper), we proposed a process placement strategy to assign processes to fixed services aiming to completely eliminate expensive network communication. Future work will address workload and resource-aware process placement to avoid resource bottlenecks and a possible increase in execution time due to concurrent execution on one server node. Furthermore, we will consider dynamic service placement to advise an optimal process to service mapping for available server nodes.

ACKNOWLEDGEMENTS

The project was funded by means of the German Federal Ministry of Economy and Technology under promotional reference "'01MQ07012"'. The authors take the responsibility for the contents.

REFERENCES

Alonso, G., Reinwald, B., and Mohan, C. (1997). Distributed data management in workflow environments. In *RIDE*, pages 82–. IEEE Computer Society.

Bauer, T., Reichert, M., and Dadam, P. (2003). Intra-subnet load balancing in distributed workflow management systems. *Int. J. Cooperative Inf. Syst.*, 12(3):295–324.

Böhm, M., Habich, D., Preissler, S., Lehner, W., and Wloka, U. (2009). Vectorizing instance-based integration processes. In *ICEIS*, pages 40–52.

Chafle, G. B., Chandra, S., Mann, V., and Nanda, M. G. (2004). Decentralized orchestration of composite web services. In *WWW*, pages 134–143, New York, NY, USA.

Habich, D., Preissler, S., Lehner, W., Richly, S., Aßmann, U., Grasselt, M., and Maier, A. (2007). Data-greyboxweb services in data-centric environments. In *ICWS*, pages 976–983.

JBoss (2010). Jboss enterprise service bus. http://jboss.org/jbossesb.

Khalaf, R. and Leymann, F. (2006). E role-based decomposition of business processes using bpel. In *ICWS*, pages 770–780, Washington, USA.

Maneth, S., Mihaylov, N., and Sakr, S. (2008). Xml tree structure compression. In *DEXA Workshops*, pages 243–247. IEEE Computer Society.

Martin, D., Wutke, D., and Leymann, F. (2008). A novel approach to decentralized workflow enactment. *EDOCC*, pages 127–136.

Ng, A. (2006). Optimising web services performance with table driven xml. In *ASWEC*, pages 100–112.

OASIS (2007). Web services business process execution language 2.0 (ws-bpel). http://www.oasis-open.org/committees/wsbpel/.

Suzumura, T., Takase, T., and Tatsubori, M. (2005). Optimizing web services performance by differential deserialization. In *ICWS*, pages 185–192.

Werner, C., Buschmann, C., and Fischer, S. (2004). Compressing soap messages by using differential encoding. In *ICWS*, page 540.