Proc. IFCIS In'l Conf. on CoopIS, Kiawah Island, South Carolina, June 1997

A Distributed Execution Environment for Large–Scale Workflow Management Systems with Subnets and Server Migration

Thomas Bauer, Peter Dadam
Dept. of Databases and Information Systems, University of Ulm
Albert–Einstein–Allee, 89069 Ulm, Germany
{bauer, dadam}@informatik.uni-ulm.de

Abstract

If the number of users within a workflow management system (WFMS) increases, a central workflow server (WF–server) and a single local area network (LAN) may become overloaded. The approach presented in this paper describes an execution environment which is able to manage a growing number of users by adding new servers and subnets. The basic idea is to decompose processes into parts which are controlled by different WF–servers. That is, during the execution of a workflow instance its execution (step) control may migrate from one WF–server to another. By selecting the appropriate physical servers (for hosting the WF–servers) in the appropriate LANs, communication costs and individual WF–server workload can be reduced significantly.

1. Introduction

Since a couple of years there has been a growing interest in using WFMS for implementing process–oriented application systems. As the benefit of such application systems increases with the number of applications being served, the number of workflow applications and WFMS users within a company will significantly grow year by year once it has started to go that way. Thus, the question arises how to manage large numbers of users (may be even tens of thousands [9]) and high volume data transfer (e.g. in conjunction with multi–media applications) within a WFMS.

Most existing systems use a central WF–server. It is easy to see that it becomes a bottleneck and will be overloaded under a high load. To reduce the load of the WF–server it can be replicated. This method can be used in combination with our approach (section 3.3) and is chosen in [2] (see section 5), for example. But there remains a bottleneck, namely the band–width of the subnet of the WF–server.

To see that a LAN really may become a bottleneck, let us perform a little numerical exercise. Let us assume that 300 users are working concurrently, each of them needing 5 minutes (= 300 seconds) in the average to perform one (workflow) step. This means, that in the average one step is executed per second. Let us further assume, that in the average 10% of all users have the appropriate role to execute a certain step. That is, this step should appear in the worklists of these users. Assuming a packet size of 100 byte, we will need approx. 4 packets for transmitting the worklist entries and respective acknowledgements, in total $30 \cdot 4 \cdot 100$ byte = 12 KB per second. If we further assume that the selection and execution of one step requires the transmission of 300 KB of input data and produces the same amout as output, then 12 KB + 600 KB = 612 KB = 4.9 megabits of pure data per second have to be transferred in average. Taking all the additional overhead into account this would already lead to an overload for a simple Ethernet–based LAN. In general, due to the potentially large number of individual messages, even very expensive high–speed LANs may become overloaded for a larger number of users.

Our approach to solve this problem is to distribute the load by using several subnets. Not every decomposition of a single LAN into subnets leads to the desired result, however. In figure 1a, for example, three subnets are used. Since all WF–servers belong to LAN 1, this subnet is burdened with the full communication load. The existence of the other subnets (LAN 2 and LAN 3) does not lead to any load reduction for LAN 1. The same problem appears, if we assume that in figure 1b WF–server S_1 has all of its clients in subnet LAN 3 and WF-server S_3 has all its clients in subnet LAN 1. In this case, both, LAN 1 and LAN 3 have to take the full communication load as if they would be in a single LAN. On the other hand, if in the scenario illustrated in figure 1b WF–server S_1 has all its clients in LAN 1, S_2 in LAN 2, and S_3 in LAN 3, then all communication may take place locally within the individual subnets. In this case we

0-8186-7946-8/97 $10.00

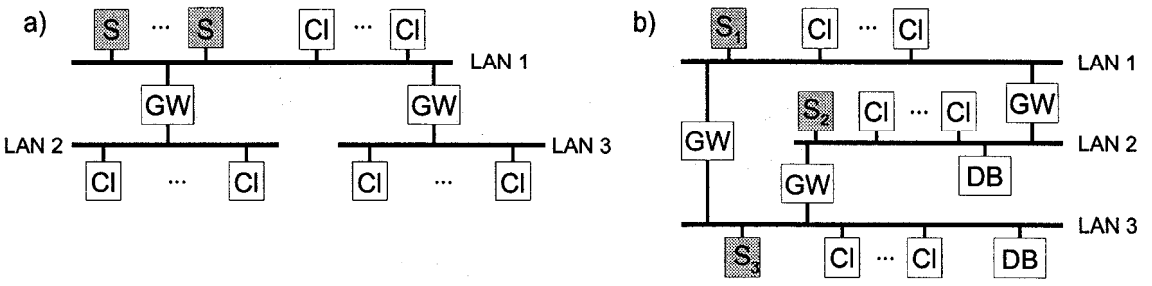

Figure 1. Structure of networks (S: WF–server with its WF–database on the same node, Cl: client, GW: gateway (router), DB: database, external data source).

achieve a significant decrease in communication load per subnet.

These examples show that the introduction of subnets may help to reduce the communication load per subnet, but it also shows that the WF–servers and clients must be in the "right" subnet to achieve the desired load reduction. As we will see later, it is not sufficient to consider workflows as a whole (each of it being controlled by a single WF–server), but that we even have to split workflows into parts each of it being controlled by another WF–server to find satisfying solutions. The development of criteria for "good" and "bad" distributions, for splitting workflows into parts, as well as the presentation of a corresponding design method for WF networks are the main issues of this paper.

One might think that load balancing should be done by the distributed system infrastructure (e.g. OSF DCE, CORBA, ...) and not by the WFMS. Unfortunately this layer has no knowledge about processes or involved roles. Hence it can not predict the communication behavior of future steps and therefore its optimizations are less effective than those of the WFMS.

The remainder of the paper is organized as follows: in section 2 the optimization problem and in section 3 the corresponding solution are described. Section 4 analyzes the efficiency of our approach, especially the creation of subnets and the decomposition of workflows. Section 5 discusses related approaches and section 6 concludes with a summary and an outlook.

2. Problem Description

The optimization problem can be sketched as follows: Given a set of processes consisting of several steps, find a distribution of processes to WF–servers such that the communication load in the subnets is minimized under the restriction that no subnet, WF–server, and gateway is overloaded. Until further notice we assume that the users (clients) can be distributed to the subnets in any way. Because this assumption is not realistic in most cases, restrictions are discussed in section 3.2.2.

The communication costs for each WF–step are caused by:

- step offering (SO_k): the transmission of the information associated with the step to all users resp. clients with an appropriate role and their acknowledgement (SO_k are the costs for maintaining the worklist of one user for step k)
- step selection (SS_k): the transmission of the information to the server indicating that a certain user has selected the step and the transmission of the the input parameters for this step to the corresponding client
- worklist refresh (WR_k): sending a delete-message for that step to the other clients to bring their worklists up–to–date and transmitting their acknowledgements
- result transfer (RT_k): the transmission of the output parameters of an activity to the server and the transmission of its acknowledgement. There is no need for a 2 phase commit protocol (2PC) to achieve transactional properties. The client stores the result in permanent storage and retransmits it if the corresponding acknowledgement times out. The WF–server ignores repeated transmissions. This is a special case of the last agent optimization of the 2PC [15]
- migration costs (MI_m): transmitting (transfering) workflow control information from one WF–server to another (see section 4.2 for details)

In our approach, the units of distribution are not complete processes, but single steps. This makes sense, because the sum of the communication costs is smallest if all steps are allocated at the optimal WF–server (and not only the

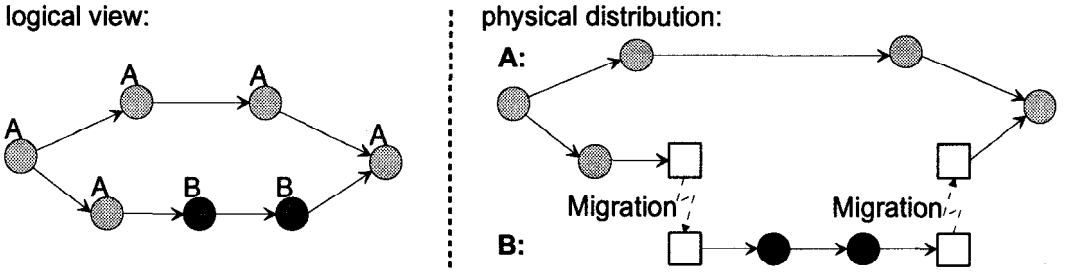

Figure 2. The steps of the process are assigned with the WF–servers A and B. The system decomposes the process in a part for WF–server A and one for WF–server B. At the points the control changes between the servers, migration steps are inserted.

process as a whole). Therefore it may become necessary to decompose a process into parts which are managed by different WF–servers as illustrated in figure 2. That is, during execution a process (resp. its control) may migrate from one WF–server to another. If this happens, all data of this process instance is copied to the subsequent WF–server and deleted at the previous one (see section 4.2 for further details).

For estimating the load of each component we make two simplifying assumptions at the moment, for a more sophisticated model we refer to section 3.1. Firstly, the execution of steps should be distributed equally during the time period T. Secondly, each user with an appropriate role should have the same probability (independent of its subnet) for selecting a step. Concerning the WF–model itself, we do not make any restricting assumptions. That is, the model may contain AND–branches, OR–branches, iterations etc. We only assume resp. require, that E_k, the number of executions of step k during a time period T, can be estimated (e.g. based on statistical information). Our approach even works with parallel branches, but (as always) they lead to a synchronization problem. We do not discuss this in detail, but we suggest the use of global process variables, because they require smaller data volumes at migration time than input/output containers [8] and at step execution time than electronic circulation folders [10]. Each variable can only be modified in one of the parallel branches, in the others it has to be modeled as read–only variable.

Let u_k denote the number of appropriate users for step k. Thus the step appears in u_k worklists. Once one user has decided to execute the step, it has to be removed from $u_k - 1$ worklists[1]. We now consider the average communication load resulting from the execution of step k. At first we look at the load $C_k^{S_k}$ in the subnet where the WF–server (S_k) resides: (F1)

$$C_k^{S_k} = \frac{E_k}{T} \cdot \Big(u_k \cdot SO_k + SS_k + (u_k - 1) \cdot WR_k + RT_k \Big)$$

The load in the other subnets can be estimated as follows. In total we have u_k users which qualify to execute step k. If u_k^x of these users (clients) belong to another subnet x (which does not contain S_k), the probability of step k to be executed in subnet x is $\frac{u_k^x}{u_k}$. If the step is executed in subnet x, the parameters have to be transmitted to that client ($SS_k + RT_k$), but there is no need to refresh its worklist ($-WR_k$). Thus the communication load for step k in subnet x can be approximated as follows: (F2)

$$C_k^x = \frac{E_k}{T} \cdot \Big(u_k^x \cdot (SO_k + WR_k) + \frac{u_k^x}{u_k} \cdot (SS_k + RT_k - WR_k) \Big)$$

The total load which a process P creates in subnet N is the sum of the step execution load and the migration load for the migration steps m. MI_m^N specifies the migration costs for step m. They are zero, if the subnet N is not affected by this migration. Thus we get: (F3)

$$C_P^N = \sum_k C_k^N + \sum_m \frac{E_m}{T} \cdot MI_m^N$$

The total load of a subnet N is: (F4)

$$C^N = \sum_P C_P^N$$

The load of the WF–servers and gateways is calculated in a similar way.

3. Derivation of Appropriate Network Topologies and Workflow Designs

The goal of this section is to develop two algorithms for calculating good distributions of the components. The first one is used for building groups of users that can be assigned to the same subnet. The second algorithm makes suggestions for a good distribution of step control to the subnets.

[1] The item can be removed from the worklist of the user that has selected this step without further communication.

3.1. Basic Idea

To minimize communication costs, the control of each step should be allocated in the subnet with the highest probability for executing this step. By doing so, the probability that all communication remains inside this subnet becomes very high. The probability for a certain step to be executed in a certain subnet can be approximated using the distribution of the users.

We introduce a weight g_i^x $(0 \leq g_i^x \leq 1)$ for each user i and subnet x which corresponds to the probability that this user chooses a step. The weight is used to describe the relative amount of time the user spends in working with the WFMS in this subnet. It is usually 1, but can be smaller if the users has only a part time job or if the user works in several subnets, for example.

The weights are used for calculating the probability that a step is executed in a certain subnet. In a WFMS only users owning one of the roles of a certain step are allowed to execute this step and the step appears only in their worklists. In our model the subnet of each user is known. With this information it is possible to calculate for each step how many "full" users exist in each subnet by computing the sum of the weights of the appropriate users for step k: $u_k^x = \sum_i g_i^x$.

3.2. Design of Processes

So far we have explained the problem and the characteristics of good solutions. In this section we show how such good distributions can be found. The problem can be solved by using a closed mathematical optimization approach, similar to some of the solutions proposed in distributed databases for finding an optimal distribution of fragment relations [11, 14, 5]. Taking this approach, one can find an optimal solution, in principle. The WF designer, however, has to specifiy a lot of (rather uncertain) parameter and constraint values which makes this approach rather expensive and thus unattractive. (Further, the practical value of the computed result is questionable, too.)

We, therefore, are in favour of an interactive and iterative approach. It starts modeling an (initial) distribution of users and WF–servers. By analyzing the resulting load, the modeling is improved iteratively and interactively until an acceptable solution is found.

3.2.1. Modeling and Analysis

In our approach, a WF design iterates through the following steps:

1. The WF designer is modeling the processes like in a central WFMS (describing the organization, data, processes; cf. upper part of figure 3).
2. The WF design system proposes an initial distribution of users and WF–servers.
3. The WF design system computes the resulting load for each component (subnet, WF–server, gateway, ...) using the model and additional statistical information.
4. If the load of all components is within the target range, the design is completed and sent to the affected WF–servers.
5. If a component is overloaded the model is modified by the WF designer using the outcome of the user distribution analysis (see "Assigning Users to Subnets") and by computing (assisted by the system) the consequences of decomposing processes (see "Distribution of Step Control"). The design process is continued at step 3.

Note, that this analysis is completely done at build time. That is, it does not disturb running processes. After WF design has been completed, the process execution model is generated, it is decomposed into parts and complemented with migration steps (if a decomposition has been selected by the WF designer), and transmitted to the affected WF–servers.

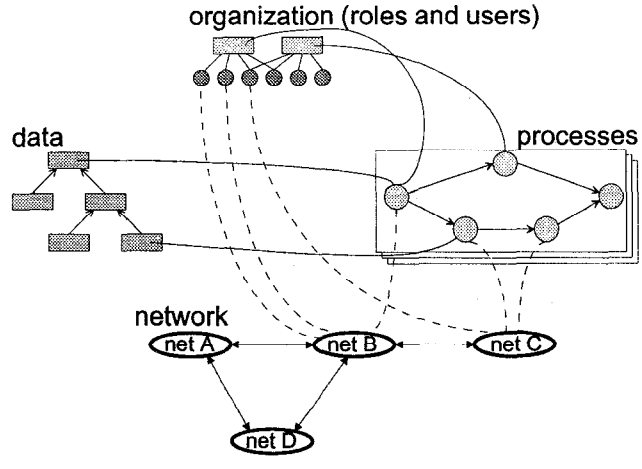

Figure 3. Modeling of processes (only partially drawn).

3.2.2 Algorithms

The WF design system proposes (initial) distributions of users and step control. The underlying analyses and algorithms are explained in the following.

Assigning Users to Subnets

The following algorithm assumes that the processes as well as the users and their roles are known. It computes "clusters" of users who can perform the same (or a similar) collection of steps. These clusters are candidates for building respective subnets. The algorithm implicitly assumes

that one starts from scratch, that is users are not yet assigned to subnets (see "Applicability" for further comments).

The problem is similar to the distribution of attributes at vertical partitioning in distributed database systems. For this reason the following algorithm is adopted from this research area [12, 13]. First of all we sketch the algorithm then we explain its meaning and the meaning of the symbols used.

1. create the user step matrix use'_{ki}
2. standardize use'_{ki} to use_{ki}
3. compute the user affinity matrix aff_{ij} using use_{ki}
4. use the known algorithms to find clusters in aff_{ij}
5. while (there is a cluster that is too large)
   decompose that cluster
6. assign the clusters to the subnets (more than one cluster per subnet is allowed)
7. change (manually) the assignment of users to subnets if necessary (and possible)

The user step matrix use'_{ki} contains the weights of the users $(g_i)^2$ with respect to their ability of executing a certain step.

$$use'_{ki} = \begin{cases} g_i & \text{if user } i \text{ can} \\ & \text{execute step } k \\ 0 & \text{otherwise} \end{cases}$$

e.g.

use'	user 1	2	3	4
step 1	1	0	0	0.5
2	0	1	1	0
3	0	1	1	0.5

Then this matrix becomes standardized so that it contains the probability that a user will execute a certain step. The divisor of the fraction will be unequal to 0, because $\forall j : use'_{kj} = 0$ would mean that no user is allowed to execute this step.

$$use_{ki} = \frac{use'_{ki}}{\sum_j use'_{kj}}$$

use	user 1	2	3	4
step 1	2/3	0	0	1/3
2	0	1/2	1/2	0
3	0	2/5	2/5	1/5

Now the user affinity matrix can be created. It contains the degree of the connection between users. If a cluster of users has high values in this matrix, they have common steps and should be in the same subnet. Such clusters of users can be found with the algorithms described in [12] and (with a better complexity) in [13].

$$aff_{ij} = \sum_k use_{ki} \cdot use_{kj} \cdot E_k$$

Frequency of step k:

k	1	2	3
E_k	100	60	10

aff	1	2	3	4
1	44.4	0	0	22.2
2	0	16.6	16.6	0.8
3	0	16.6	16.6	0.8
4	22.2	0.8	0.8	11.5

In this example one cluster would consist of the users 1 and 4 and another of the users 2 and 3. Each of these clusters could be allocated in one subnet of their own. The algorithm does not take into account the "quality" of the clusters. That is, clusters may be suggested which are too large and thus would lead to a high load in the respective subnet. In such cases clusters have to be decomposed manually into appropriate parts (subnets) to achieve the desired result. If there are more clusters than subnets, several small clusters must be assigned to one subnet, and if the physical location of a user prevents him from being in the proposed cluster, it has to be assigned to another cluster.

[2] There is no upper index for g, because until now no subnet is assigned to this user.

Applicability

This simple algorithm presented here, assumes that the WF design is starting from scratch, that is users are not yet assigned to subnets. But even if it is used in an existing WFMS environment where users have already been assigned to subnets, the results can give valuable suggestions for improvements concerning the choice of WF–servers and the decomposition of processes.

Distribution of the Step Control

As already mentioned above, workflows are assigned to WF–servers at the granularity of single steps instead of complete workflows. The calculation of the optimal distribution of the step control would have exponential complexity, because every step can be controlled by each WF–server. Our greedy algorithm discussed below will not always find the optimal solution, but it will deliver a good result for the common cases and has polynomal complexity.

The idea is to select at first the optimal subnet for every step without considering the migration costs. Then for each single step in a subnet it is checked if it is cheaper to save the migration costs (MI_m) and to assign the step to the WF–server of the step preceeding resp. succeeding the current one, with higher costs for step execution (SS_k, RT_k) and worklist maintenance (SO_k, WR_k). This is also done for all groups of 2, 3,... steps controlled by one and the same WF–server. The algorithm can be sketched as follows:

```
for each step: assign the WF–server of the subnet with
the most appropriate users
for i = 1, 2, ...
  for each group with i steps in the same subnet:
    check if it is cheaper to control that group of steps
    by the WF–server before or after the current one
      if yes: assign the group to that WF–server
```

The algorithm can be easily extended for the case that some steps are pre–assigned to specific WF–servers (e.g. because of organizational restrictions) and cannot be assigned to any other WF–servers. These steps are assigned to the dedicated WF–server (independent of the costs resp. the distribution of the users) and marked as "locked steps". The algorithm does not consider to reassign them.

3.3. Refinements

Our method achieves scalability by distributing the steps of the processes among the WF–servers. If there is only one relevant process with only one step, however, it is not possible to distribute anything, because one step can only be controlled by one WF–server. Even though this is not a typical scenario for a WFMS, there are several solutions (besides using future hardware and/or future networks):
One solution consists of splitting the process into several processes. If a process has to serve customers, one could e.g. assign the customers with names A ... M to process P_1 and N ... Z to another process P_2. P_1 and P_2 can then be controlled by different WF–servers.
Another solution is to extend our approach with WF–server replication. Instead of one WF–server for each step, several servers are used in different subnets. Only one of them may be in the optimal subnet, however, the others have to be in less suitable subnets. But even in this case the load can be reduced (see section 4.1), if it is distributed equally among the WF–servers. This is possible e.g., by randomly choosing one of these servers for starting or migrating the processes.

The external data sources[3] shown in figure 1 are a performance–critical aspect, too. They may also become a bottleneck. Therefore they have to be taken into account during the analysis. For this purpose the amount of communication with them has to be estimated for each step. It can be reasonable (where applicable), to use several (independent) databases in order to keep communication local to a subnet as often as possible.

4. Efficiency Analysis

In the subsequent two sections we will analyze the communication costs in different scenarios. At first we consider the case that processes are not decomposed, i.e., no process migration takes place. This means, that all steps of a process are controlled by one and the same WF–server. Subsequently we will analyze scenarios with process migration.

4.1. Using Multiple Servers without Process Migration

In the following, we analyze the communication traffic in the subnets which is caused by the maintenance of worklists (SO_k, WR_k) and the transfer of parameter data (SS_k, RT_k). We assume that the processes do not migrate (i.e. they are controlled by one WF–server from their beginning until their termination) which approximates also the case that migration costs are small when compared with the step execution costs.

In the sequel, we analyse three interesting cases. Some related cases are mentioned, others can be easily derived in the same way.
Case 1: All clients are located in the subnet of the corresponding WF–server.
Case 2: The majority of the clients is located in the "right" subnet.
Case 3: The majority of the clients is located in the "wrong" subnet.
For each of these cases we compare two scenarios. In the first scenario we have only one subnet with one central WF–server. In the other scenario we have two subnets, each with a WF–server, connected by a gateway. This is a scenario as described in section 1. The load in the subnets would be exactly the same, if there would be more than one WF–server in each subnet. For reasons of simplicity we consider only two subnets, but the results would be the same for any number of subnets.

To simplify our analysis we ignore the weights of the users for a moment and assume that every client having the appropriate role has the same probability for executing the step. Since the data exchanged with each client has the same volume (because SO_k, SS_k, WR_k, RT_k are equal for all users), there is no need for counting messages or data packets. To compare the load in the different subnets, it is sufficient to count how many clients are involved in the execution of how many steps for a given set of processes. Figure 4-1a illustrates the case where two processes, each involving two clients (discriminated by solid and dotted lines), are executed by one server. In this case we count 4 "connections" in this subnet in total. Opposed to that, two servers, each executing one process, are used in figure 4-1b. In this case we have only 2 "connections" in each subnet.

To simplify the comparison, we assume that the total number of processes to be served is equally distributed among the servers. This leads to equal loads for all WF–servers and subnets and it becomes possible to compare the subnet load in the two scenarios.

Case 1: Each client is located in the same subnet as the WF–server of the corresponding steps. In this case the gateway need not to be used and – as each WF–server serves 50% of the processes – the communication in each subnet is halved. This is the best case because all communication is completely taking place inside the subnets. Figure 4-1a shows that in the scenario with one LAN there are 4 connections between the WF–server and clients, while in figure 4-1b there are only 2 connections per subnet. If there were n subnets instead of two, the load would be $\frac{1}{n}$th of the not distributed case.

Case 2: Here, the majority, but not all clients are in the "right" subnet. It is evident that this also leads to an im-

[3] Only data required outside the WFMS is stored in external data sources, because there are more possibilities for optimizations (e.g. migration) for data stored in the (local) database of the WF–server.

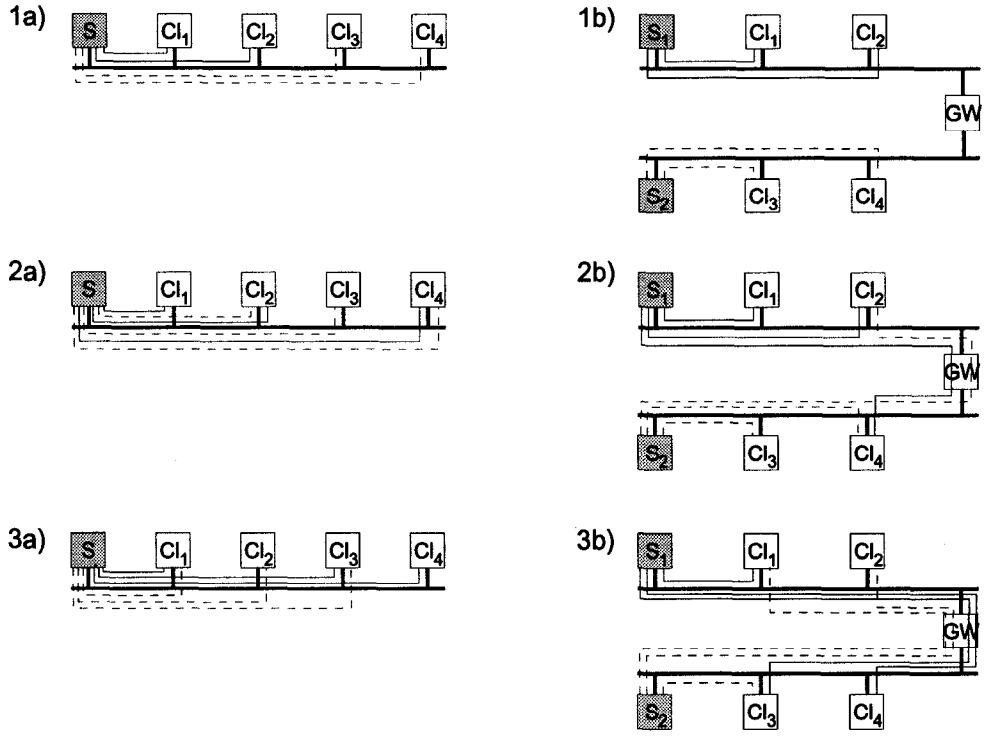

Figure 4. Comparison of scenarios with one and two subnets for different communication patterns.

provement compared to having only one net (see figure 4-2). There are 2 clients in the subnet of the WF–server and 1 client in the other one. This leads to 4 connections instead of 6.

If there were as many clients in the subnet of the WF–server as in the other one, we would save 25% of the communication. One can demonstrate this in figure 4-2 by deleting the connection to Cl_1 and Cl_3. In this case, there remain 4 (2a) resp. 3 (2b) connections. The saving of 25% is achieved because in the case of a communication with a client in the own subnet (50% of the cases) the other subnet (50% of the subnets) is not used.

Case 3: Here most of the clients are located in the "wrong" subnet. Even in this case we save communication as demonstrated in figure 4-3. In this example only 1 client is in the subnet of the WF–server but 2 clients are in the other one. Even in this unfavorable scenario only 5 connections are needed instead of 6. The saving exists as long as there are clients in the subnet of the WF–server, because for communication with these clients the other subnet is not used.

The worst case is that all clients are in the "wrong" subnet. Even in this case, however, the subnet load is not higher than in the single net version. All subnets are used for all communications, but this case should never occur in practice. One always could distribute the processes in another way, so that at least for some steps there is a user in the "right" subnet. For more than two subnets in a completely intermeshed network there are always subnets with a reduced load. Even if all clients are in "wrong" subnets there are always subnets which are not involved in communication for certain steps because there is no through traffic. Therefore our approach is also eligible for unfavorable distributions of users.

To achieve the best efficiency the users and the control of steps must be distributed in such a way that as many users as possible are in the same subnet as the WF–server that controls their steps.

4.2. Using Multiple Servers with Process Migration

In this section we consider the case that – due to the distribution of users – no single WF–server does really optimally fit for controlling a given process. In such cases it may be better to decompose the process into parts such that each of it can be controlled by the optimal WF–server. That is, we consider the case that after execution of a set S_1 of steps in subnet N_1, the process control is migrated to another WF–server in subnet N_2 which controls the execution

of the remaining set S_2 of steps. The crucial question is whether the reduction in communication load (due to the migration) is counterbalancing the migration costs. We can discriminate three cases:

- ideal case
- mixed positive case
- negative case

Ideal case: In this case, after migration all clients are in the same subnet (N_2) as the new WF–server. Thus, after migration, subnet N_1 has no communication load for the remaining steps any more and the load in subnet N_2 is the same as without migration. Given the load resulting from the migration ($C_{MI} = \frac{E_m}{T} \cdot MI$) – which we have to count two times because it occurs in both the "sender" and the "receiver" subnet – and using formula (F2) we can compute, how many remaining steps ($\rightarrow S_2$) are needed to make a migration rewarding. This is the case, if the following unequation is satisfied:

$2 \cdot C_{MI} < \sum_{k \in S_2} C_k^{S_k}$

where $\sum_{k \in S_2} C_k^{S_k}$ describes the savings in subnet N_1, if the control for the steps $S_2 = \{s_{2_1}, \ldots s_{2_n}\}$ is migrated to the WF–server in the other subnet.

Mixed positive case: In this case, after migration most – but not all – clients related to the remaining steps ($\rightarrow S_2$) are in the "right" subnet ($u_k^{N_1} < u_k^{N_2}$). This means that we achieve some saving in subnet N_1 because some of the communication load is now completely handled in subnet N_2. In subnet N_2, however, the load is higher than without migration, because this subnet is now also burdened with the communication to the clients in subnet N_1. The decision problem can be formulated as follows: The migration is rewarding if the following unequation holds: (F5)

$$2 \cdot C_{MI} < \sum_{k \in S_2} \Big(\underbrace{(C_k^{S_k} - C_k^{N_1})}_{\text{savings in } N_1} - \underbrace{(C_k^{S_k} - C_k^{N_2})}_{\text{additional load in } N_2} \Big)$$

$$\Leftrightarrow 2 \cdot C_{MI} < \sum_{k \in S_2} \underbrace{(C_k^{N_2} - C_k^{N_1})}_{(*)}$$

As we consider here the case that $u_k^{N_1} < u_k^{N_2}$, expression $(*)$ will always be positive. That means, having enough steps in S_2, migration is rewarding.

Negative case: In this case, after migration most clients are in the "wrong" subnet ($u_k^{N_1} \geq u_k^{N_2}$). The analysis is the same like in the previous case and thus also leads to the same unequation (F5). In this case, however, expression $(*)$ can never become positive and thus migration is never rewarding (as expected).

The circumstances under which migration is rewarding are demonstrated in the following numerical example: We assume equal steps in S_2 with the same frequency for each step (E_k) and for the migration step (E_m). They shall have the following characteristics:

input parameter volume:	IN_k	$= 300$ KB
output parameter volume:	OUT_k	$= 100$ KB
total process instance data:	$INST$	$= 1000$ KB
data transmitted for a worklist entry:	WL	$= 0.1$ KB
data transmitted for an acknowledgement (minimal packet size):	Ack	$= 0.1$ KB
appropriate users in subnet N_1:	$u_k^{N_1}$	$= 10$
appropriate users in subnet N_2:	$u_k^{N_2}$	$= 200$
users altogether:	u_k	$= 210$

Based upon this information, values for the variables in the formulas in section 2 can be calculated as follows:

$SO_k = WL + Ack = 0.2$ KB
$SS_k = WL + IN_k = 300.1$ KB
$WR_k = WL + Ack = 0.2$ KB
$RT_k = OUT_k + Ack = 100.1$ KB
$MI = INST + Ack = 1000.1$ KB

Now (F1) can be used to calculate the load in the subnet in which the WF–server is located:

$C_k^{S_k} = \frac{E_k}{T} \cdot (210 \cdot 0.2 + 300.1 + 209 \cdot 0.2 + 100.1)$ KB
$= \frac{E_k}{T} \cdot 484$ KB

If the WF–server is located in N_2 the resulting load in subnet N_1 can be calculated with (F2):

$C_k^{N_1} = \frac{E_k}{T} \cdot 23$ KB

If the WF–server is located in the "wrong" subnet N_1, the load in N_2 is much higher:

$C_k^{N_2} = \frac{E_k}{T} \cdot 461$ KB

Using formula (F5) we can calculate that the migration is rewarding if

$\frac{E_m}{T} \cdot 2 \cdot 1000.1$ KB $< \sum_{k \in S_2} \frac{E_k}{T} \cdot (461 - 23)$ KB

With $E_k = E_m$ follows ($|S_2|$ is the number of steps in S_2):

$2000.2 < |S_2| \cdot 438$

Therefore the migration is rewarding if there are at least 5 steps in S_2.

The algorithm calculating a distribution of step control (section 3.2.2) does not directly use unequation (F5), but it compares the costs with and without migration. A process is decomposed only if the migration is rewarding. In this section we have shown that there exist cases in which a migration is rewarding.

5. Related Work

The approach described in this paper concentrates on the reduction of communication load in large–scale WFMS environments. To achieve this goal, we use subnets as well as the decomposition and distributed control (via process migration) of processes. To the best of our knowledge, there are no other approaches that deal with the reduction of communication load in a WFMS. Hence process instance migration was never used for this purpose. For our type

of application scenario we are only interested in process–oriented systems (as opposed to e.g. Lotus Notes), because in large scale environments the corresponding functionality is needed. Most process–oriented systems use a central WF–server and are therefore not (directly) suitable for our target environment. In the following we discuss some distributed approaches.

FlowMark [7, 8] is a system with a central WF–server, but it is possible to execute a "subprocess" in another FlowMark system (local domain). If process control shall be distributed, the concept of subprocesses has to be used, because only subprocesses can be executed at remote servers. The logic is comparable to a remote procedure call. That is control returns to the caller after completion of the subprocess.

Exotica [2] uses so called "clusters" to achieve parallelism. A cluster consists of one WF–database and replicated WF–servers. The user has to connect with one WF–server of each cluster. By replication, load reduction is achieved for the WF–servers within a cluster. The control of a process instance stays in the cluster in which the process was started. By selecting an appropriate cluster, load balancing among the clusters can be achieved, but it may cause long distance communications to the users.
In MOBILE [6] server replication is used, too. The WF–model is separated into several perspectives (organization structure, control flow, etc.) each with its own database and its own server. If one of these servers is overloaded it becomes replicated. Static data of these servers are replicated, dynamic are partitioned and assigned to only one server. Scalability is achieved under the assumption that there exist independent partitions (e.g. for different departments). Both approaches (Exotica and MOBILE) do not consider subnet load. Therefore, process instance migration is not used.

There are several approaches which do not use WF–servers at all. They have in common, that after finishing a step the process instance migrates directly to the node of the following step. Usually a reliable communication system is used for this purpose. The disadvantage is that the role resolution is only done at the first time when a step becomes available [1]. Consequently, this step is not offered to users which are connecting at a later point in time. In INCAS [4] every step is dedicated to exactly one user. Thus, there is no need for synchronization, but the functionality is very limited. A similar approach is used in the Mentor project [16, 17], where each step is associated with exactly one "entity". To each entity belongs a WF–server, but it is only responsible for its local clients. Thus there is also no global role resolution. Opposed to this, in Exotica/FMQS [3] a step is offered to all users with an appropriate role. Because there exists no WF–server for coordinating the step selection, a distributed (and therefore expensive) synchronization mechanism has to be applied. But the problem remains, that a step is not offered to users which connect to the WFMS after this step is ready for selection. All these approaches have in common that at every step the whole process instance migrates.

6. Summary

In this paper we have concentrated on the aspects of how to optimize the communication load in WFMS environments with a large number of users. We have shown that with the usage of subnets and by assigning WF–control to the "right" WF–servers, the load can be distributed and thus more users can be served. We have described how – based on easily to obtain information – one can develop algorithms for calculating such distributions.

We have further shown that it can be favourable to not always treat and control workflows as a whole but to decompose them into parts which are controlled by different WF–servers. We have analyzed under which circumstances such a "process migration" is rewarding.

There are further possible improvements of our approach, e.g. dynamic optimization of the step control distribution at runtime. Furthermore several aspects as the dynamic structural changes of processes at runtime and the handling of the resulting exceptions or process abortion with compensation of already executed steps have to be integrated in our approach. This will be subject of our future work.

References

[1] G. Alonso, R. Günthör, M. Kamath, D. Agrawal, A. E. Abbadi, and C. Mohan. Exotica/FMDC: Handling Disconnected Clients in a Workflow Management System. In *Proc. of the Third Int. Conf. on Cooperative Information Systems*, pages 99–110, Vienna, May 1995.

[2] G. Alonso, M. Kamath, D. Agrawal, A. E. Abbadi, R. Günthör, and C. Mohan. Failure Handling in Large Scale Workflow Management Systems. Technical Report RJ9913, IBM Almaden Research Center, Nov. 1994.

[3] G. Alonso, C. Mohan, R. Günthör, D. Agrawal, A. E. Abbadi, and M. Kamath. Exotica/FMQM: A Persistent Message–Based Architecture for Distributed Workflow Management. In *Proc. of the IFIP Working Conf. on Information Systems for Decentralized Organisations*, Trondheim, Aug. 1995.

[4] D. Barbará, S. Mehrotra, and M. Rusinkiewicz. INCAS: A Computational Model for Dynamic Workflows in Autonomous Distributed Environments. Technical report, Matsushita Information Technology Laboratory, Princeton, May 1994.

[5] P. Dadam. *Verteilte Datenbanken und Client/Server–Systeme (Distributed Databases and Client/Server Systems).* Springer–Verlag, 1996. (in german).

[6] P. Heinl and H. Schuster. Towards a Highly Scaleable Architecture for Workflow Management Systems. In *Proc. of the*

7th Int. Conf. and Workshop on Database and Expert Systems Applications, DEXA'96, pages 439–444, Zürich, Sept. 1996.

[7] IBM. *FlowMark – Installation and Maintenance, Version 2 Release 2*, Second edition, Feb. 1996. Document Number: SH12-6260-00.

[8] IBM. *FlowMark – Modeling Workflow, Version 2 Release 2*, Second edition, Feb. 1996. Document Number: SH19-8241-01.

[9] M. Kamath, G. Alonso, R. Günthör, and C. Mohan. Providing High Availability in Very Large Workflow Management Systems. In *Proc. of the 5th Int. Conf. on Extending Database Technology*, pages 427–442, Avignon, Mar. 1996.

[10] B. Karbe, N. Ramsperger, and P. Weiss. Support of Cooperative Work by Electronic Circulation Folders. In *Conference on Office Information Systems, IEEE Computer Society*, pages 109–117, 1990.

[11] H. Morgan and K. D. Levin. Optimal Program and Data Locations in Computer Networks. *Comm. of the ACM*, 20:315–322, 1977.

[12] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou. Vertical Partitioning Algorithms for Database Design. *ACM Transactions on Database Systems*, 9(4):680–710, 1984.

[13] S. Navathe and M. Ra. Vertical Partitioning for Database Design: A Graphical Algorithm. In *Proc. of the 1989 ACM SIGMOD Int. Conf. on Management of Data*, volume 18, pages 440–450, Portland, June 1989.

[14] M. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1991.

[15] G. Samaras, K. Britton, A. Citron, and C. Mohan. Two–Phase Commit Optimizations in a Commercial Distributed Environment. *Distributed and Parallel Databases*, 3(4):325–360, Oct. 1995.

[16] J. Weissenfels, D. Wodtke, G. Weikum, and A. Kotz–Dittrich. The Mentor Architecture for Enterprise–wide Workflow Management. In *Proc. of the NSF Workshop on Workflow and Process Automation in Information Systems*, pages 69–73, Athens, May 1996.

[17] D. Wodtke, J. Weissenfels, G. Weikum, and A. Kotz Dittrich. The Mentor Project: Steps Towards Enterprise–Wide Workflow Management. In *Proc. of the 12th IEEE Int. Conf. on Data Engineering*, pages 556–565, New Orleans, Mar. 1996.