

Excercise Sheet 6

Solution

Lecture Distributed Systems

Winter Term 2025/26

Exercise 1: Programmierung: JNDI

- a) JNDI (Java Naming and Directory Interface) provides methods with standard directory operations (reading, writing, deleting, adding attributes to objects, and searching for objects based on these attributes). In addition, it defines a unified interface for accessing various directory services.

Java applications provide two ways to work with names and directory services using JNDI. One is the traditional way - traditional in this context means to access actually existing directory services (file system, mail, etc.). The second possibility is to use the directory as a storage location and an organizational form for storing objects.

- b) You will find the solution to this problem in in the archive [106eFiles.zip¹](#) on the lecture's web page.
- c) You will find the solution to this problem in in the archive [106eFiles.zip²](#) on the lecture's web page.

Exercise 2: Load balancing strategies

- a) Static load distribution: a schedule is created before execution; disadvantage: all processes and their execution times must be known a priori, dynamics of processes and external load cannot be taken into account.

Dynamic load distribution: processes are (only) placed on a node when they are created and remain there; disadvantage: later change of the load situation (due to dynamics of the processes or external load) is not taken into account.

Preemptive dynamic load distribution: processes can switch nodes during their runtime (even several times); disadvantages: migration problems, thrashing possible; advantage: very adaptive, especially with external load.

- b) Not all ready processes cause the same amount of load (e.g. processes that frequently wait for I/O). A better load measure could be the CPU load (i.e. how much % of the time the CPU is actually calculating), which however is not always easy to determine and can also fluctuate strongly. Memory requirements, page error rates, I/O rates, communication rates, etc. could also be included in load metrics. However, the (frequent) determination of these metrics means additional overhead, which is not always amortized by the "better" metric. Experiments have shown that (due to this overhead) usually no better load balancing is achieved by such metrics (i.e. the runtime of a program system is not reduced).

Exercise 3: Graph partitioning and list scheduling

Load distribution by graph partitioning:

Given is a number of concurrent, communicating processes and the communication load between each two processes. This is shown as a graph. The nodes represent the processes and the edges show the communication load. Objective: to distribute the resulting load as evenly as possible in order to prevent overload on individual nodes. The graph is partitioned in such a way that the weight sum of the cut edges is minimal, i.e. the communication between nodes is minimized. All load balancing strategies base their decisions on some *load metric*.

¹<http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/vs/106eFiles.zip>

²<http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/vs/106eFiles.zip>

List-Scheduling:

The program consists of several dependent tasks, which may work on output data of their predecessor tasks, but do not communicate *during* processing. This is represented as a DAG. The nodes contain tasks with execution times and the edges show the necessary communication along with the transmission time. Objective: To schedule the tasks so that the processing time (*makespan*) is minimized.

Similarities:

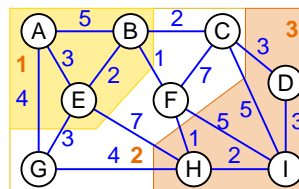
Graphs with tasks as nodes and communication as edges, edge weights, task structure is fixed, number of CPUs is given, schedule (assignment of processes to CPUs) is calculated before program start, goal is usually to minimize the runtime, ...

Differences:

Program structure (with graph partitioning: concurrent processes that communicate during execution; with list scheduling: tasks that may require output data from other tasks but do not communicate *during* execution), meaning of node weights, undirected vs. directed graph, ...

Exercise 4: Load Distribution via Graph Partitioning

The new distribution looks like this:



In this distribution node 1 has the processes A, B, and E, node 2 has the processes C, F, and G, and node 3 has the processes D, H and I. The cut between node 1 and 2 contains the edges AG, EG, BF and BC with a communication load of $4+3+1+2 = 10$. The cut between node 2 and 3 contains the edges CD, CI, FI, FH and GH with a communication load of $3+5+5+1+4 = 18$. The cut between node 1 and 3 contains the edge EH with a communication load of 7. The sum is $10 + 18 + 7 = 35$.