SIMULATING DISTRIBUTED APPLICATIONS IN AN ACTIVE NETWORK

Tomas Koutny and Jiri Safarik

University of West Bohemia, Faculty of Applied Sciences
Univerzitni 8, 306 14 Plzen, Czech Republic

txkoutny@kiv.zcu.cz (Tomas Koutny)

Abstract

Continuous development of network architectures emphases two related characteristics: dynamic protocol deployment and utilization of free resources. Active networks present such network architecture. Unlike tradition networks, packets are superseded by capsules, which contain a custom code performing specific activities, each time a capsule visits a node. The dynamic protocol deployment involves a custom code injection at remote nodes and its subsequent execution to implement desired behavior of the network. As the custom code executes, it consumes resources such as processor, memory, bandwidth, etc. In this paper, we present a simulator of active network in use with a computation-intensive distributed application and a heterogeneous hardware. The simulator provides a virtual active network, where no application-specific behavior is coded into the network and each node closely models a behavior of a scheduler of a real, non-simulated, operating system. Into such network, we inject a virtual distributed application and observe utilization of resources available in the network. It is possible to enter a number of parameters, which affects the size of simulated network including the variety of used hardware, behavior of the distributed application comprised of thinking and waiting times, code branching probabilities, communication, migration rules and random number generators. As an example, we give an output of the simulator with input parameters, which we measured on a real, non-simulated, distributed application.

Keywords: active, network, simulator, distributed computing, heterogeneous environment.

Presenting Author's Biography

Tomas Koutny. He started his PhD on Faculty of Applied Sciences at University of West Bohemia, at the Department of Computer Science and Engineering. Nowadays, he has a full time contract at the department. His research activities are oriented toward distributed systems. His PhD thesis addresses the load redistribution in a heterogeneous environment. His current research focuses on active networking, development of active server, deployment and development of new services and protocols. This paper represents a part of his current research.

ISBN 978-3-901608-32-2

1 General

Recently, we undertook a research project on load-redistribution in a heterogeneous distributed environment [1, 2]. As an implementation platform, we used active network. To verify behavior of proposed method, we developed a simulator of computation-intensive distributed application in an active network. There are number of options, which could be used to set specific characteristics of the network and the application, so it is possible to use it for a simulation within other research projects.

In traditional networks, all packets have fixed header and payload. The header stores information such as destination address and routers use this information to forward or discard individual packets. On contrary, the packet in the active network [3, 4], called capsule, is associated with a particular code that runs every time a capsule visits a node. The node that is capable to run capsule's code is an active node. This code may perform various tasks such as custom routing, network management, injecting of applications or collecting specific information such as performance snapshot.

Active application is a process that runs at the active node and uses capsules to transfer data among the nodes. A distributed application consists of possibly many active applications. The active application has to use capsules for its migration, while the active node directly supports the capsule's migration.

Each capsule or an active application is isolated from others within active node's execution environment, where it runs. Nevertheless, it is possible to use primitives, such as global state for instance, for a communication between active applications and capsules at a single node. As an active application or a capsule runs, it consumes resources. As a resource, processor time, memory, bandwidth, time to live, etc. is considered.

Second section discusses simulation of active network, while the third section presents its parameters. Next, we give overview of simulator's architecture and implementation in section four. Fifth section provides simulation results, which can be obtained using the simulator. Section six states the conclusion and outlines future work.

2 Simulation

The intention is to create a simulation, which reflects current architectures of operating systems, execution environments and distributed applications in the environment of active networks. The active network simulation is based on our reference implementation.

The simulator conforms to following needs:

- It provides a virtual active network, no application-specific behavior is coded into active nodes.
- Network addressing and routing is the same as in the reference implementation.
- Each active node is programmed to behave like a real node running applications without any priority changes to benefit the distributed application.
- Active applications and capsules are described with a code that reflects the reference implementation; some portion of the code is even shared.
- The virtual active network and injected distributed application are configurable via input parameters.
- Simulation results are acquired from states of nodes, not from the state of a running distributed application; the state of the network is periodically sampled and visualized.
- Each node of simulated network provides the same features as the reference implementation. This applies mainly to the scheduling of capsules, active applications, resources, global states and routing tables.

The network-addressing scheme is the same as in the reference implementation including the routing. The simulator uses the grid topology, because a significant number of various topologies can be mapped onto it. Applications accessing routing table of particular node see node's neighbors only like in the real network.

The code that simulates the real active node manages a list of applications and capsules running there. There is API implemented at each simulated node, which provides access to node's services – for instance the access to underlying network. In addition, the node allocates resources to applications and capsules to control their runtime. The node picks up no particular application or capsule for a performance boost.

There are two approaches to discrete-time simulation [5]: event interpretation and pseudo-parallel processes. Main part of the event-interpreted simulation is an event handler, where all events are generated and processed. This usually leads to a non-trivial complex event handler and a centralized approach as the handler acts as a main controlling entity. Since we simulate network, where each node is a standalone entity executing active applications and capsules, we chose the pseudo-parallel processes approach.

2.1 Scheduler

In the real, non–simulated, operating system, the main component determining an execution of particular processes is the scheduler. Therefore, the main component of our simulated node is the scheduler too. In the discrete time simulation, the time is measured with an integer variable. When some entity simulates computation, so called thinking time, it is suspended

and then resumed on a given discrete time. While it may be enough for a general simulation of processes scheduling, we decided rather to follow the real behavior of a scheduler.

The real processor is able to execute a fixed amount of the same set of instructions per given interval. Similarly, our simulated node is able to execute a fixed amount of virtual instructions per single step of the discrete simulation time. This way, we specify a computational performance of the simulated node. As a simplifying condition, we assume that overhead of scheduler, and the rest of system processes, is covered within the performance of simulated node. According to our reference implementation, each active application and each capsule runs in its own execution environment. The scheduler is responsible for allocation of processor-time quanta [6, 7] to these environments. Operating system of the simulated node maintains a context for each running execution environment. Beside information such as environment's state, this context contains a number of virtual instructions to be executed – expressing the length of thinking time.

Thus, instead of specifying a fixed discrete time, for which an entity has to remain suspended, we assign a fixed number of virtual instructions to be executed on behalf of a given process/environment. With the advance of simulation time, each node decreases this number of virtual instructions. The maximum, by which it can decrease this number, is determined by the performance of the node. Thus, the number of instruction for execution is not decreased for all environments at a particular node.

Moreover, the scheduler may reorder the schedule of execution environments to ensure that all environments execute in a given number of recent steps of the simulation. This leads to possible race conditions just like in real operating system. The total time of an execution of a single environment depends on the total number of execution environments at the node, not on a fixed number only. Therefore, when the simulation entity goes to thinking mode, the exact time, for which it would remain thinking, cannot be set. When the thinking time is over, the node calls a pre-defined method of the simulation entity that makes the decision on entity's life.

2.2 Communication Clusters

Usually, only a subset of all processes of a distributed application communicates together during an application's runtime [8, 9]. We call such subset a communication cluster [10]. There is at least one communication cluster per distributed application, or their number is equal up to the number of processes. Generally, they may change dynamically, but from the design of distributed applications, we know that this does not occur often, if ever, during the runtime.

For each process, the simulator picks a number of processes in order of their creation. This way, the simulator forms communication clusters.

2.3 Node API

As a real operating system exposes its services to applications via a set of predefined application

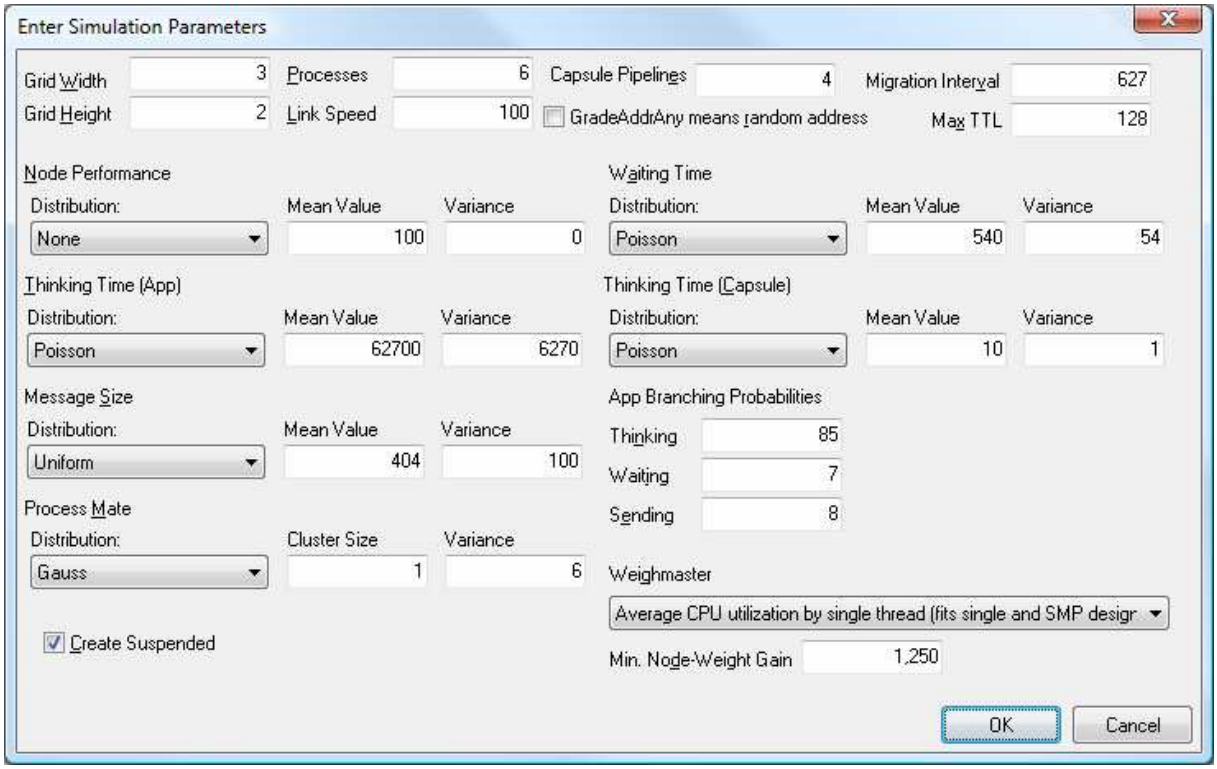

Figure 1. Simulation Parameters

programming interfaces (API), our simulated node does the same to simulated entities, which act as active applications and capsules. API of simulated node covers following areas:

- Environmental access – access to the simulated network, routing information
- Access to simulation – generators of random numbers, the simulation time
- Caching – for a limited time, storing of data standing out of the application/capsule address space
- Rendezvous – allowing capsules and applications to meet and coordinate their actions
- Capsule manipulation – modification of header, data load abstraction
- Control operations – forward and discard operations over capsules, injecting of capsules and applications, change of the entity state

Functionality of APIs provided to applications and capsules is classified this way:

- Functionality available to application as well as to capsules
- Functionality available to applications only
- Functionality available to capsules only

3 Parameters

To affect the behavior of the execution environments as well as applications and capsules, we introduced following simulation parameters – see Fig.1.

3.1 The underlying distributed environment

- Width and height of the grid – simulated nodes are connected into a regular grid, because almost any topology can be mapped onto it by declaring particular nodes as virtual
- Link speed – bandwidth of connections between two nodes in full duplex
- GradeAddrAny – during the initialization of the simulation, processes of distributed application are injected to this address; this can be either one node to simulate start of the application, or it can be random address to simulate an already running application.
- Maximum TTL – time to live for all capsules
- Capsule pipelines – number of capsules, which can run simultaneously; lower number means a shorter execution time of a single capsule, but higher number means more parallelization
- Node performance – number of virtual instructions, which a simulated node can execute in a fixed time interval

3.2 Application-specific features

- Processes – number of processes of a distributed application
- Thinking Time
- Waiting Time
- App Branching Probabilities – probabilities, with which a simulated application will perform computing, thinking, waiting for an event or sending message after a previous action
- Process Mate – a cooperating process in a communication cluster

3.3 Capsule-specific features

- Thinking time – interval a capsule spends in mode, which simulates active processing such as routing at give node; an analogy to thinking time of an application
- Message size – size of a data message sent from one process to another; the size affects the time that a capsule spends in the node, while passing through

3.4 Specific features of the load-redistribution method

- Migration interval
- Weighmaster – approaches to the evaluation of nodes' performance:
  - Average CPU Load
  - Average number of applications
  - Average load of individual threads (takes unused CPU into account)
  - Average load of individual threads extrapolated for remote node (SMP design - SMP stands for symmetric multi-processor systems)
  - Average CPU utilization by single thread (fits single and SMP designs)
- Minimum node-weight gain – based on a performance snapshot of a given node, its performance weight is computed as a real number; to evaluate a node as a possible migration target, its performance must be at least equal to the performance of the local node multiplied by this parameter

ISBN 978-3-901608-32-2

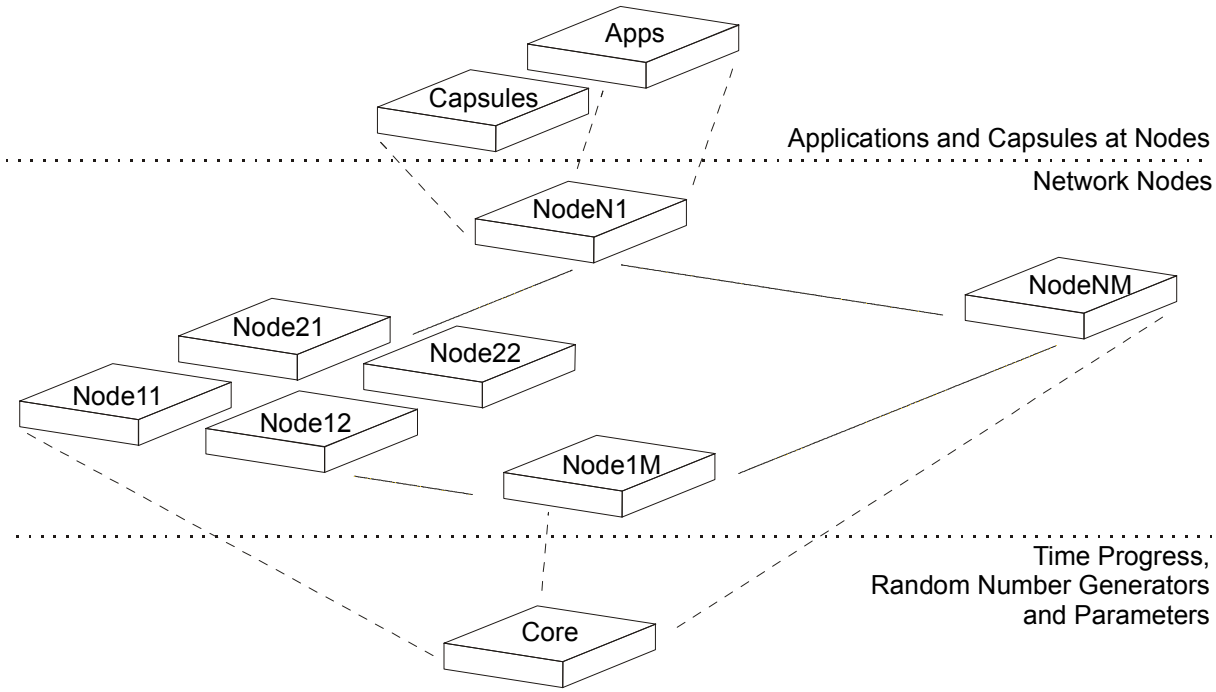

Figure 2. Executive Part of the Simulator

There are four types of probability distributions, which are offered for generating the random numbers:

- None – a constant is always returned
- Uniform – each value has the same probability
- Poisson – Poisson distribution
- Gauss – Gaussian distribution

"Mean Value" and "Variance" parameters are used to set valid parameters for a chosen distribution.

4 Architecture and Implementation

As depicted in Fig. 2, there are three hierarchy levels in the simulator's architecture:

- Core
- Network Nodes
- Applications and Capsules

The core of the simulator is responsible for the global simulation time and for handling the common functionality such as random number generators. It maintains the set of simulated network nodes, which compose the grid. The core calls the "life" method of every node in each step. The "life" method is firstly called for each node, and the node calls the method for capsules and applications, which run at the given node.

Simulated network node is responsible for:

- Maintaining a list of applications and capsules running there via calls to their "life" methods
- Performing scheduling of applications and capsules
- Running applications and capsules
- Keeping connections to other nodes
- Providing services to applications and capsules
- Encapsulating access to services provided by the core such as a random number generator
- Providing functionality described in section 2

Applications and capsules carry out operations, which simulate the run of a distributed application. They create message flow and utilize virtual processors – i.e. they consume available resources. As they run on the top, all other entities provide services to them. However, the rest of entities, nodes and core, affect their runtime.

Firstly, the user interface thread is created to get simulation parameters from the user. Subsequently, it creates an executive thread, running the executive part described above, and passes the parameters, so that the core can create the simulation entities. In the first place, it creates nodes and connects them. Then, it injects pre-defined number of applications, which will later generate capsules. Finally, the core runs the simulation.

The entire simulation application runs in three threads:

- User interface – processing of the input from the user and provides the output to the user
- Executive part – main simulation described above
- Visualization – periodically samples state of the executive part and creates a visualization, which is presented via the user interface

Entire simulation terminates on a user request received via the user interface thread.

5 Simulation Results

For the demonstration of the simulator, we pick a set of parameters that corresponds to our reference implementation, in active network, of a parallel prefix sum computation [11], which uses synchronization primitives and its processes communicate together.

```
var a[1:n]:int
    sum[1:n]:int
    old[1:n]:int
    d:int=1

//initialize elements of sum
sum[i:1..n]::
  sum[i]:=a[i]

barrier
{SUM: sum[i]=a[i-d+1]+…+a[i]}

while d<n do

  //save old value
  old[i]:=sum[i]
  barrier

  if (i-d)>=1 then
     sum[i]:=old[i-d]+sum[i]

  barrier
  d:=d*2

end while
```

The parameters were obtained from a log of the reference implementation ran on Intel Core Duo 1.86GHz. Link speed corresponds to 100Mb/s link at full duplex. Fig. 3 depicts the network. VI stands for Virtual Instructions, while ST stands for a simulation time unit – i.e. the simulation step. The execution speed 100VI/ST corresponds to Intel Core Duo 1.86GHz. Migration interval expresses minimum work needed, before a migration may occur.

Application Specific Parameters

Working Time	Poisson, 62700 VI
Waiting Time	Poisson, 540 VI
Working Probability	85%
Waiting Probability	7%
Sending Probability	8%
Processes	12
Migration Interval	627 VI

Capsule Specific Parameters

Working Time	Poisson, 10 VI
Size	Uniform, 304B – 504B
Link Speed	100B/ST

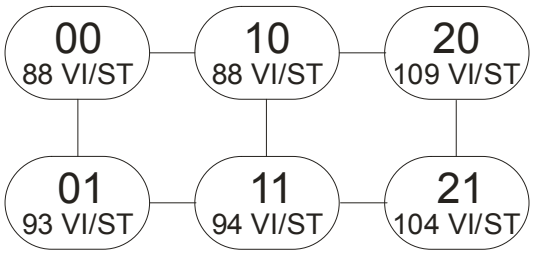

Figure 3. Testing Network

We have used Poisson distribution of probability as the measured data conform to it. There is a similar work aimed at the performance analysis in a heterogeneous environment [12], which uses the same distribution for simulation.

The simulator records following per node:

- Time progress
- Number of active applications
- Number of capsules
- Processor time consumed by applications
- Processor time consumed by capsules
- Total processor time available per simulation step
- Network traffic – stored internally only

Fig. 4 to 9 depict the load of individual nodes in a testing network during the runtime of distributed applications. The primary process was injected at node 11, where it injected remaining processes, which spread into the network.

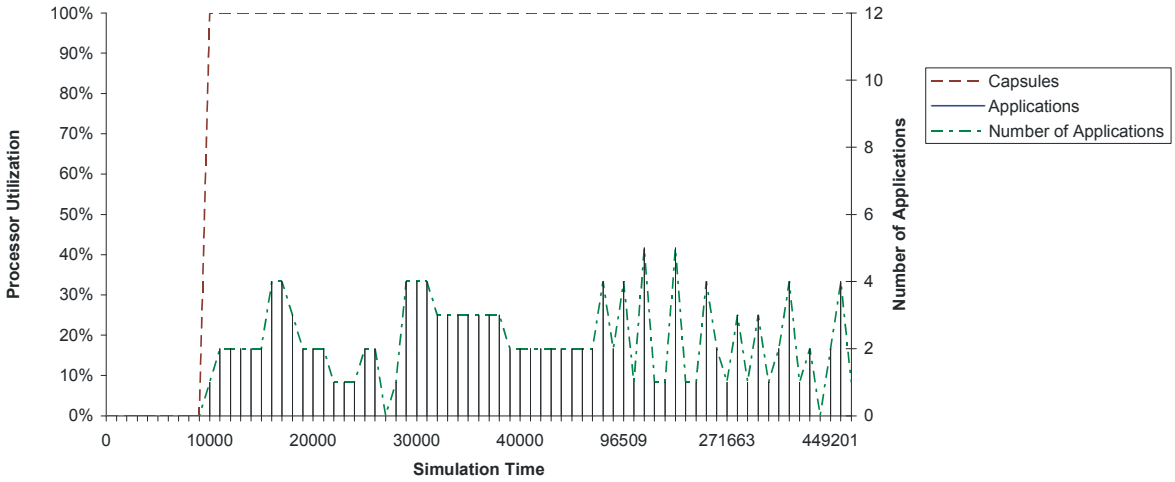

Figure 4. Node 00

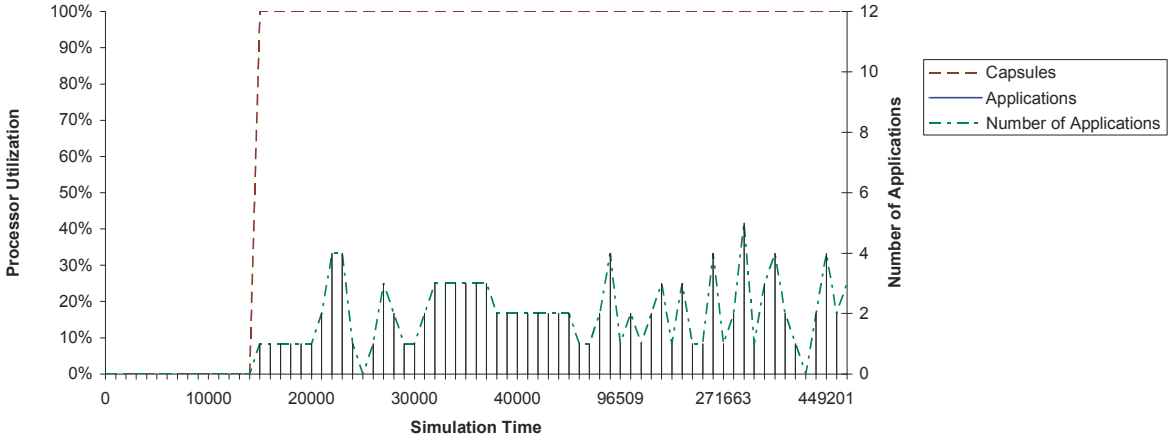

Figure 5. Node 10

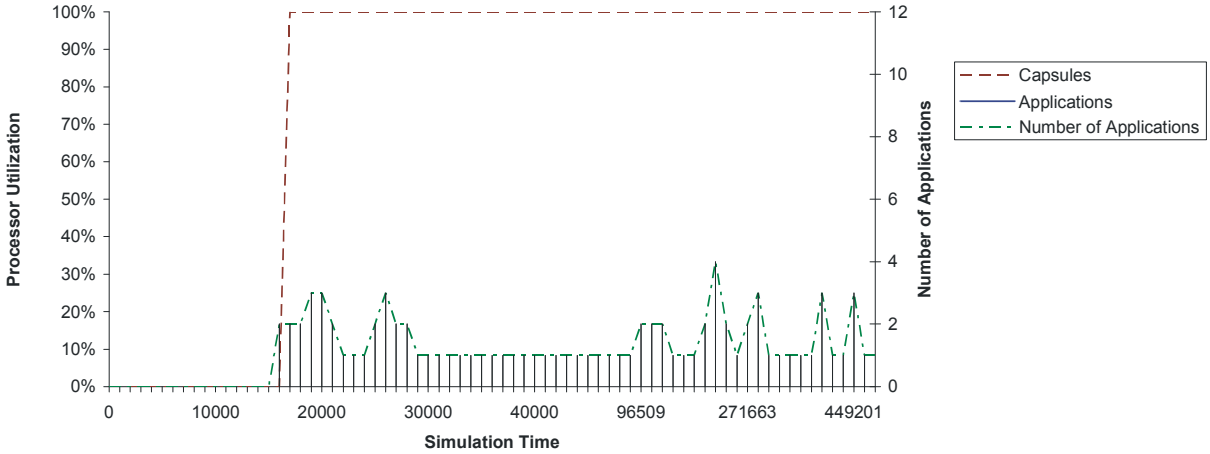

Figure 6. Node 20

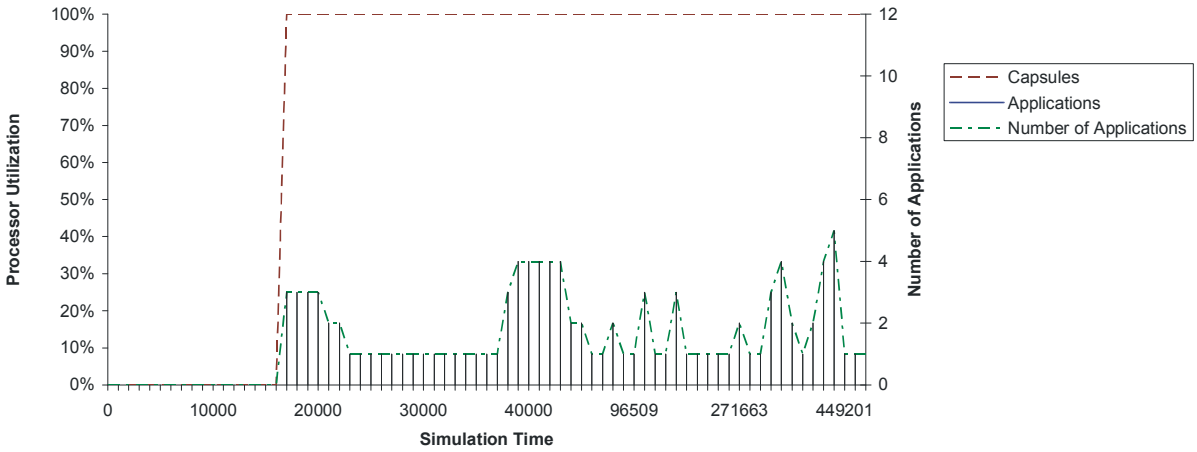

Figure 7. Node 01

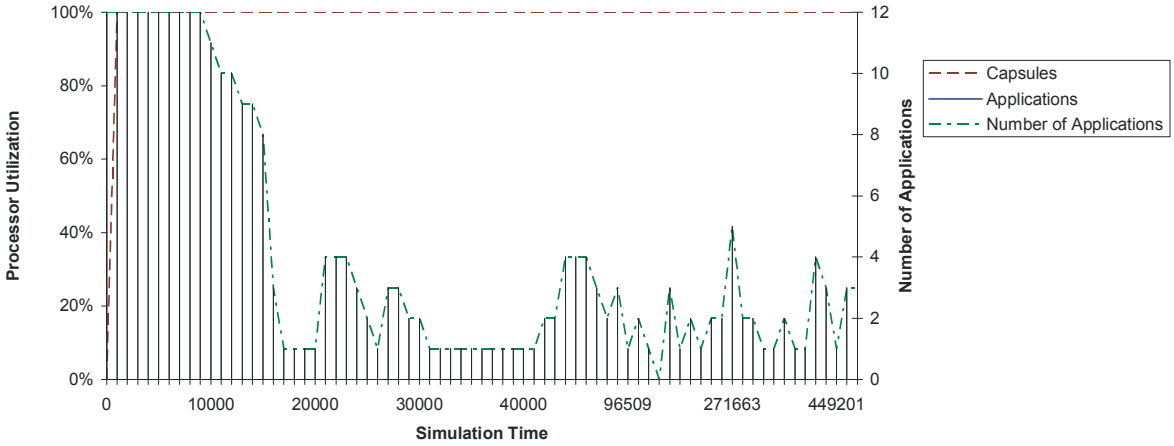

Figure 8. Node 11

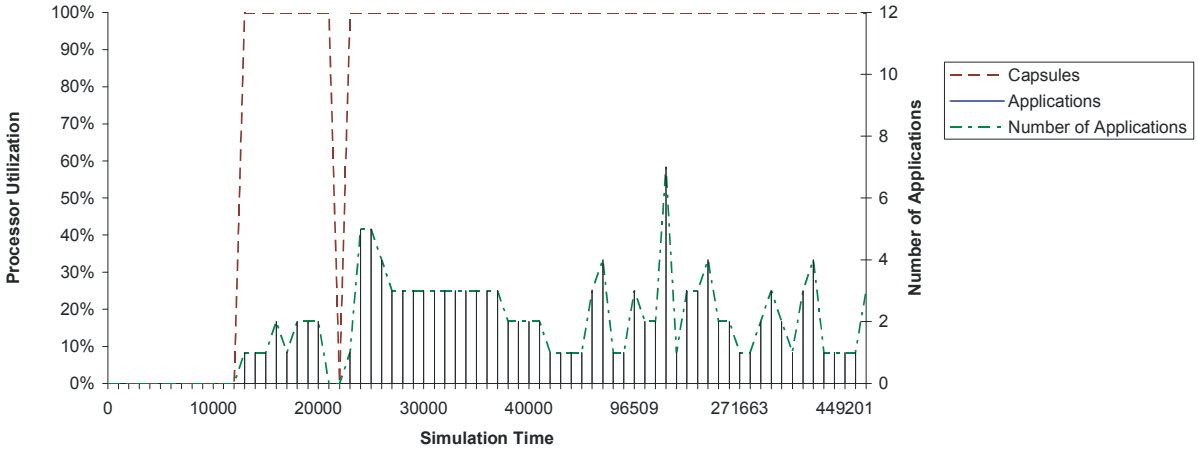

Figure 9. Node 21

Active applications have not migrated right at the beginning, as there was no guarantee that it will result into better a usage of available resources. Later, when it seemed profitable, the migrations started.

During the runtime, we can see several peaks in the number of processes running at particular nodes. This relates to the fact that all nodes are overloaded and the load redistribution method tries to enhance the performance by reorganizing active applications placement to reduce the communication delays.

The simulator randomly generates a virtual topology even during the runtime, because we were interested in a response to such changes. This is another cause of the peaks in number of processes. As it is apparent from presented graphs, it was possible to prevent mass migration of the majority of all processes.

6 Conclusion and Future Work

We have designed and implemented a simulator of active network running a generic model of a distributed application with the load redistribution algorithm.

Using the simulator, we are able to predict a behavior of a distributed application on almost any network topology and then track down the usage per a single node. As the examples, we can mention the study of new network protocols and load prediction algorithm [13].

Future work would include a network topology editor along with extending the scope of output statistics to e.g. individual capsules and active applications.

7 Acknowledgment

The work was sponsored by the Ministry of Education, Youth and Sport of the Czech Republic - "University spec. research - 1311".

8 References

[1] T. Koutny and J. Safarik, “Gradient Method with Topology Discovery for Load-Balancing in Active Networks”, 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04), Brno, Czech Republic, 2004, pp. 75 – 85

[2] T. Koutny and J. Safarik, “Load Redistribution in Heterogeneous Systems”, The Third International Conference on Autonomic and Autonomous Systems,, Athens, Greece, 2007

[3] D. L. Tennehouse and D. J. Wetherall, "Towards an Active Network Architecture", Proceedings of the DARPA Active Networks, Conference and Exposition (DANCE.02), 0-7695-1564-9/02

[4] AN Node OS Working Group, "NodeOS Interface Specification", January 10, 2001

[5] R. M. Fujimoto, “Parallel and Distributed Simulation Systems”, Wiley, 2000

[6] D. Solomon, “Inside Windows NT”, Microsoft Press, 1998

[7] D. P. Bovet and M.Cesati, “Understanding the Linux Kernel; 2nd Edition”, O’Reilly, 2002

[8] G. R. Andrews, "Foundation of multithreaded, parallel, and distributed programming", Addison Wesley, 1999

[9] N. Lynch, “Distributed Algorithms”, Morgan-Kaufman, 1997.

[10] T. Koutny and J. Safarik, ”Maintaining Communication Channels for Migrating Processes in the Environment of Active Networks”, Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks - PDCN 2005, Innsbruck, Austria, 2005, pp. 100 – 106

[11] G. R. Andrews, “Concurrent Programming: Principles and Practice”, The Benjamin/Cummings Publishing Company, Inc., 1991

[12] B. Javadi, M. K. Akbari and J. H. Abawajy, “Performance Analysis of Heterogeneous Multi-Cluster Systems”, 2005 International Conference on Parallel Processing Workshops (ICPPW'05), Oslo, Norway, 2005, pp. 493 – 500

[13] S. F. Bush and A. B. Kulkarni, “Active Networks and Active Network Management – A Proactive Management Framework”, Kluwer Acedemic/Plenum Publishers, New York, 2001

