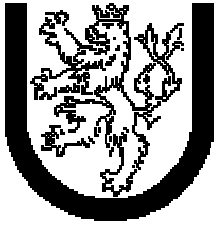
Evaluating Performance of Nodes in Distributed Environment of Active Networks

Tomáš Koutný



University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering
Plzeň, Czech Republic

DSS Version 1.0; txkoutny@kiv.zcu.cz; April 18, 2005

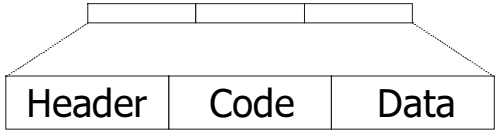
Evaluating Performance of Nodes in Distributed Environment of Active Networks

- Objective
- Slight Introduction into Active Networking
- Distributed Environment
- Communication
- Load-Balancing Method
- Testing Application
- Runtime Results
- Simulation
- Math Proof
- Conclusion

Objective

- Method of evaluation of node performance for use in a load-balancing method
- Dynamic, decentralized and adaptive approach
- Usage with active-networking enabled applications

Slight Introduction into Active Networking



- Packet can contain executable code, or reference to, in addition to standard data load
- Each network node is OS to packet
- Execution Environment executes packets' code to perform custom activities such as custom routing
- Distribution and execution of the code is base feature of active networking
- Packets are used to inject new/refine existing functionality into/of the network

Distributed Environment

- Own execution environment, codename Grade
- Inspired by ANTS/BEES
- Intended for computation-intensive applications
- Protocols such as TCP and SCTP are preferred for capsule transmission over ANEP/UDP to minimize time needed to execute capsule's code
- Objects such as nodes, capsules, applications and global states are identified with GUIDs
- Performance monitor measures resources' usage

Communication

- Ideal cluster concept; identifies together communicating applications
- Decentralized and node-independent addressing
- Originally, applications were organized into ring
  - Each application held reference to 2 neighbors
  - Transactions
  - Overhead increase when many neighbors changed node
- Now, ring was left to advantage of lazy updates
  - Application migrates to new node, leaves trace at old node and sends notification to all applications in cluster after it migrated

Load-Balancing Method
The Idea Behind

- Based on observation of flowing of liquids
- Network is a container with varying depth
- Depth is given by performance of individual nodes
- Liquid in the container presents running applications and capsules => as they traverse/migrate/create/terminate through/in the network, they cause waves
- Waves about given threshold cause app. migration
- Goal is to achieve still surface

Load-Balancing Method
Performance Scouts

- Each application is responsible for node evaluation; it periodically sends capsules, called performance scouts, into network
- Scouts are not sent, when not all available processor time is consumed
- When scout gets to node, it sends node's performance snapshot to application and splits itself to probe node's neighboring nodes
- Link speed is measured by time needed to get particular performance snapshot; includes all delays

Load-Balancing Method Performance Snapshot

- The method is supposed to run on SMP systems
- Performance snapshot includes
  - Benchmark of single processor
  - Number of processors
  - Average usage of all processors
  - Average lowest usage of single processor
  - Average number of running execution environments
  - Average available size of physical memory
- Performance monitor is able to track utilization of processor time by particular application

Load-Balancing Method Reaction Time

- Reaction time is affected by
  - Frequency of sampling the state of resources
  - Length of interval that is covered by single snapshot
  - Interval after which application sends scouts
- Longer intervals are more resistant to fluctuations
- Shorter intervals better copy actual state of resources

Load-Balancing Method
Node Performance

- Floating point number; higher value means higher performance
- General performance function:

$$node\ performance = \frac{available\ processor\ time}{link\ speed\ between\ nodes}$$

- So far, the best performance function is:

$$local\ node's\ performance = \frac{used\ \ processor\ time \cdot benchmark}{link\ speed\ between\ nodes}$$

$$remote\ node's\ performance = \frac{(1 - lowest\ usage\ of\ \ one\ \ processor) \cdot benchmark}{link\ speed\ between\ nodes}$$

Load-Balancing Method
Migration Candidate

- Nodes with performance higher than local node are considered as candidates for migration
- Minimum multiply difference presents affordable costs
- Final candidate is chosen as a match of all candidates again nodes of applications in cluster
- Local leader election prevents migrations, which would harm overall performance
- For instance, mass migration of all applications to one another node is prohibited by local leader

Testing Application

- Parallel computation of prefixes
  (input: 1, 2, 3, 4, 5; output: 1, 3, 6, 10, 15)
- Computation uses barriers for synchronization
- No central process/active application coordinating others
- No static assignment of processes/active applications to given nodes

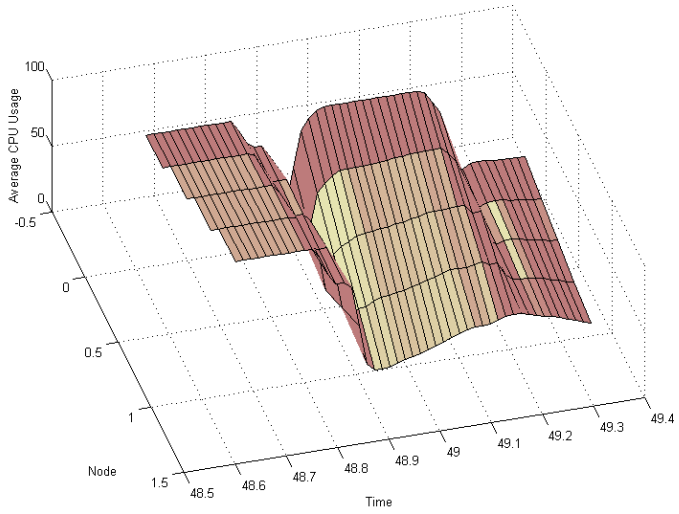
Testing Application

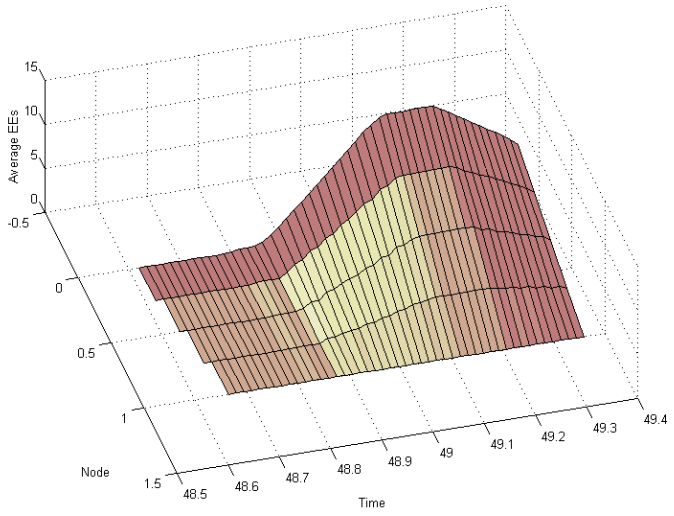
```
var a[1:n]:int, sum[1:n]:int, old[1:n]:int
sum[i:1..n]:: var d:=1
  sum[i]:=a[i]                #initialize elements of sum
  barrier
  {SUM: sum[i]=[a-d+1]+…+a[i]}
  do d<n ->
    old[i]:=sum[i]            #save old value
    barrier
    if (i-d)>=1 -> sum[i]:=old[i-d]+sum[i] fi
    barrier
    d:=d*2
  od
```

Testing Application

- The algorithm is faster than serial one, when used in multi-threaded application on multi-processor system with shared memory only
- Obtaining temporal results computed by another process requires additional MOVS usage regardless of IPC
- Timing of MOVS, ADD, FLD, FST and FADD instructions causes Grade application to spend most of their run time suspended, when waiting for a synchronization event
- Used to test method to properly recognize need for migration
- Processor wasting during adding to satisfy migration

Runtime Results
No Processor Wasting

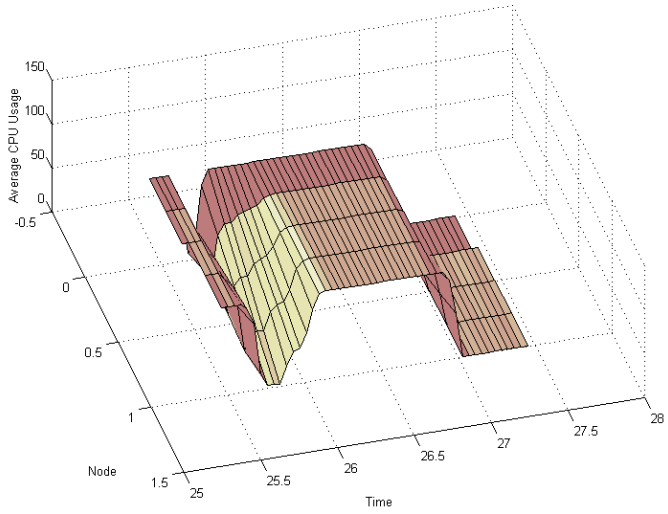




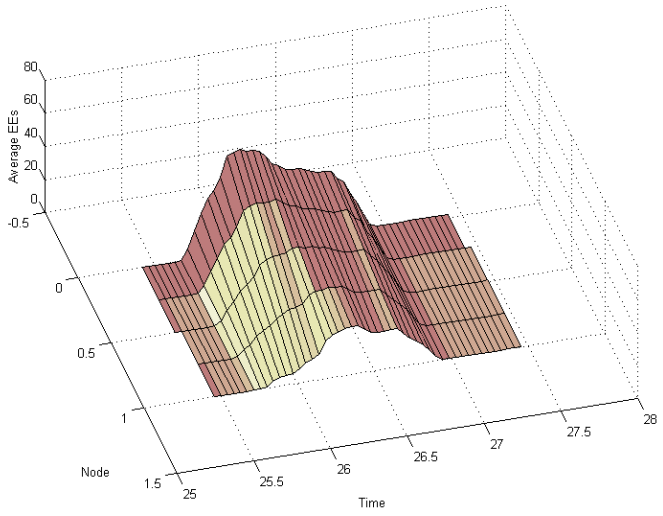
Average CPU Usage

Average Number of EEs

Runtime Results
With Processor Wasting

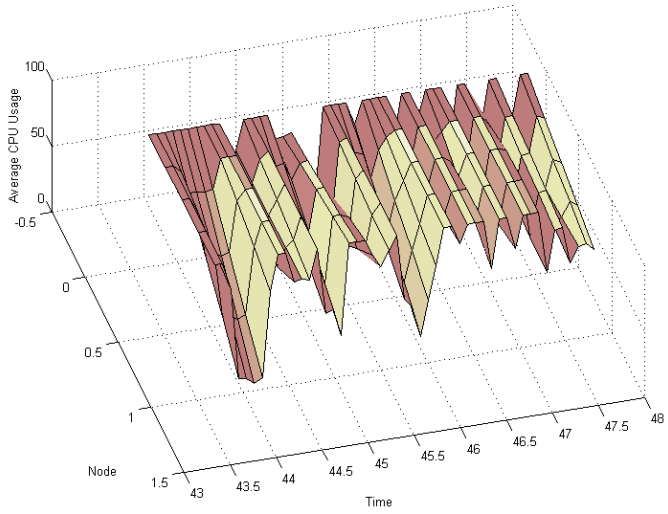


Average CPU Usage

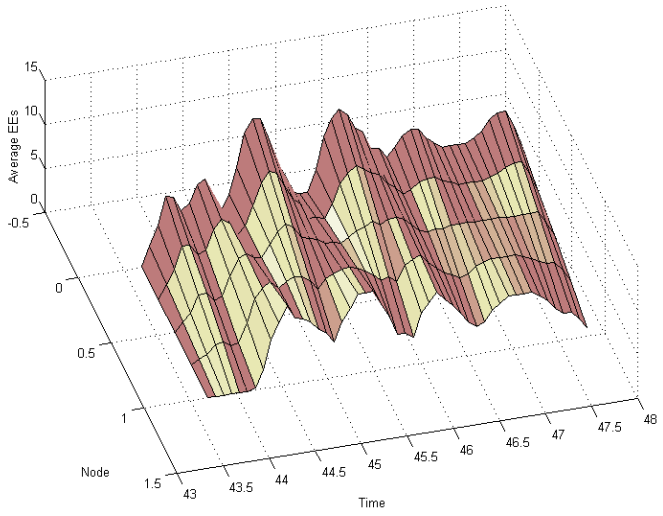


Average Number of EEs

Runtime Results
With Processor Wasting; Detailed Look



Average CPU Usage



Average Number of EEs

Simulation

- Developed simulation of active network
- Grid topology is used as any other topology can be mapped onto it by declaring certain nodes and links as virtual with zero processing delays
- Each node is treated as a single processor system
- Each node has its own task scheduler simulating traditional scheduler of operating systems
- Simulator can operate under several different options affecting behavior of network and applications
- Live simulator show is coming now...

Math Proof

- There is a proof stating that for a very large number of very small tasks, applications, their distribution, as evenly as possible, leads to shorter completion time
- To distribute tasks evenly means to have every processor loaded with one task at least, to assure that each task has as much of processor time as possible available for its execution
- Task is just a set of instructions consuming processor time up to 100% of time given by processor's power in a time interval

Math Proof

- Let us call ideal processor time consumption the amount, which is consumed by one task per processor not running any other task
- Then, until sum of these consumptions of all applications executing on a given processor is lower or equal to 100%, the processor is not overloaded and communication overhead is reduced, if messages do not have to flow through network
- This is exactly how the proposed load-balancing method treats utilization of processors in the network; in addition it pursues reduction of comm. overhead by keeping processes close to each other

Conclusion

- The method is able to correctly detect, when to migrate applications, and to find better nodes to boost overall performance
- Dynamic, decentralized and adaptive manner
- Simulation and runtime results conform to each other; the math proof guarantees the minimization of entire completion time of distributed application

- There is an article with the same title that gives the same topic in more detailed look, but excluding the math proof