New tool for performance estimation of the block data processing algorithms in high-load systems

25 April 2013

V.Bashun, V.Minchenkov, A.Povalyaev

Goals and Problem Statement

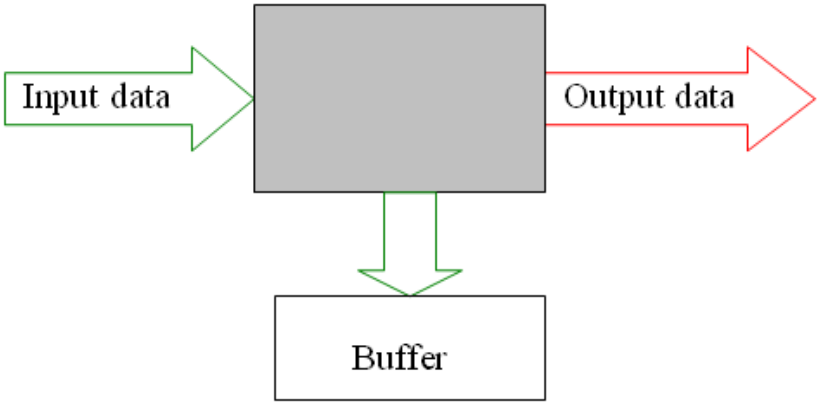
- GOAL:
  - Estimation of performance of block data processing by encryption/compression algorithms and gathering the statistical information for variant parameters of the environment
- It is often not possible to modify existing algorithms (compression, crypto, etc.) or even it’s implementations. Meanwhile, it's quite promising to try to optimize the environment and the way in which there algorithms are run.
- The architecture of the operating system (Linux) contains a set of parameters through which you can tune it up and adjust the working environment of a particular data processing thread.

Existing tools

- A variety of programs for estimation and profiling Linux kernel exists
  - Measuring performance of software-hardware platform
  - Profiling software implementation
- The goal of new tool is to measure the performance of block data processing algorithms
  - Like compression or encryption
  - Running in OS kernel

Compared Algorithms

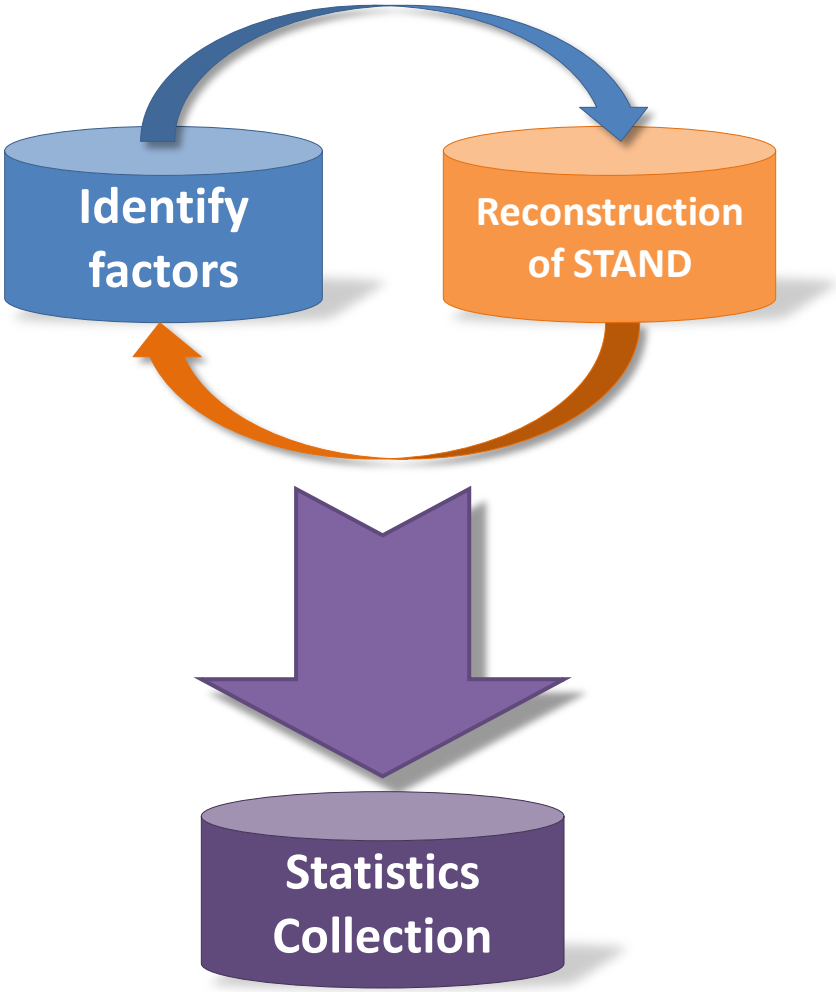
- Block data processing algorithms with known interfaces and closed implementation are given (black box).



- Empirical estimation of influence of environment on the efficiency of given algorithms.

Key Stages of Work

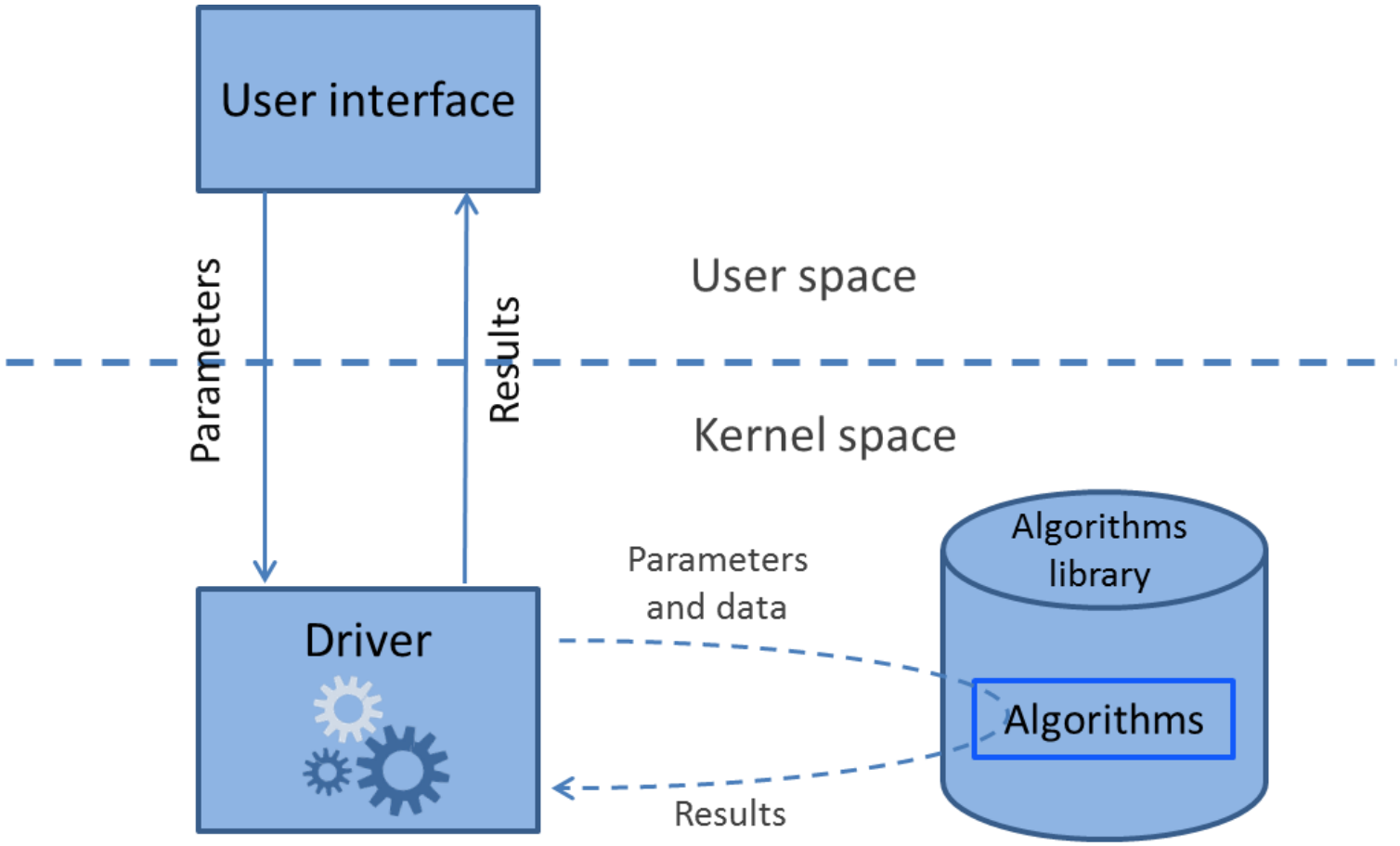
1. Identification of possible factors influencing the performance of algorithms. Kernel parts being examined:
   - scheduling tasks and process management
   - memory management
2. Building a test system that allows automated evaluation of the influence factors on the algorithms
3. Statistics collection and identification of critical factors
4. Algorithms estimation & comparison



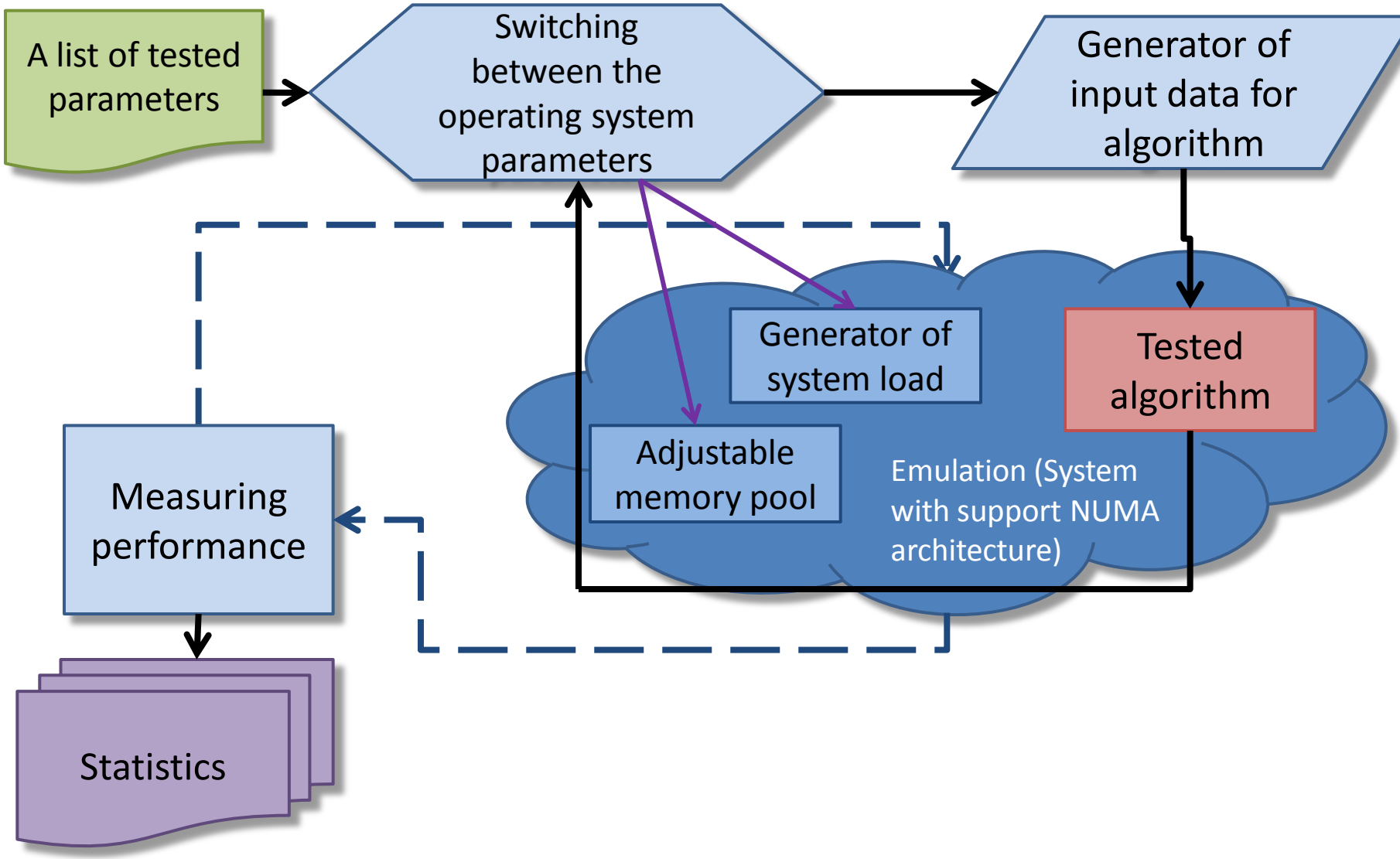
Requirements

- Ease of adoption of new algorithms
  - Using abstraction layer with known API
- Easy to use
  - Setting plan of experiment using simple configuration file
  - Running as easy as calling executable file
- Possibility to introduce new factors

Architecture of tool

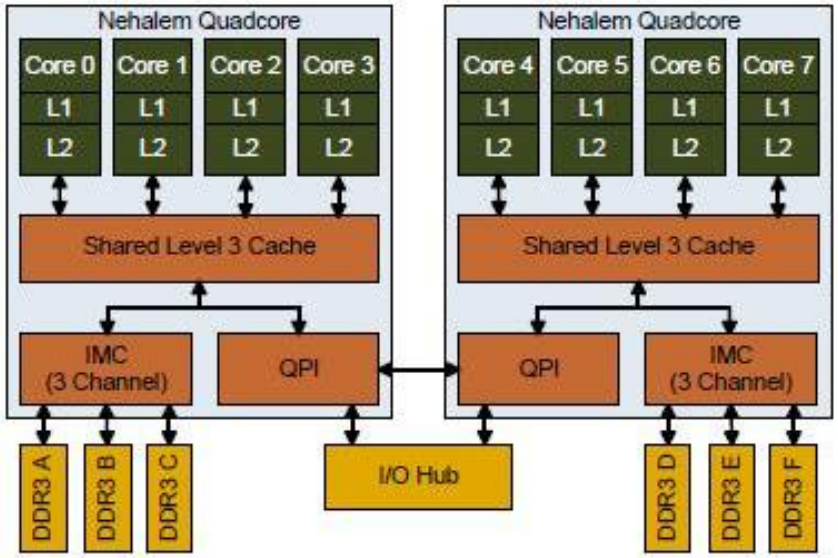


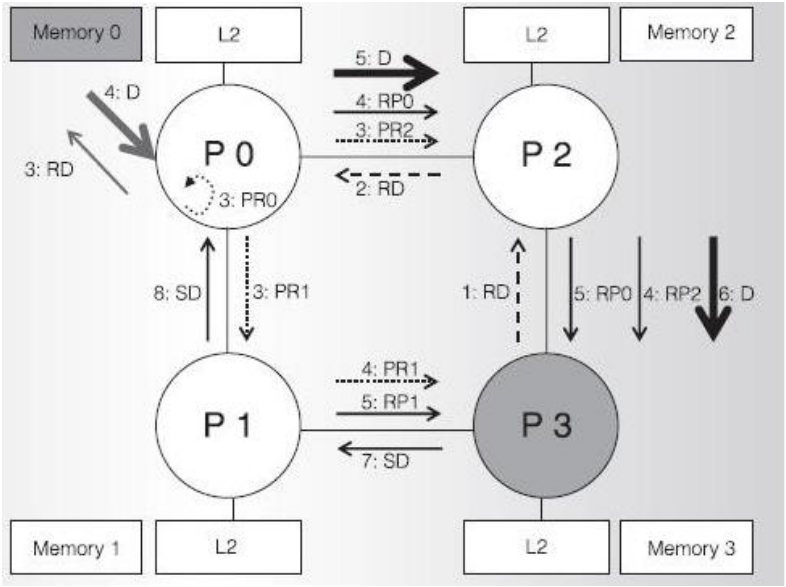
Preliminary Model Stand for Tests



NUMA architecture

- Non-Uniform Memory Access
  - RAM access time depends on it's location with respect to processor
- Supported by most modern platforms:
  - Intel Nehalem EP (Xeon 5500);
  - AMD Opteron;
  - IBM POWER6/7;
- Supported by operating systems
  - Linux, starting from kernel v.2.6;
  - API for applications (libnuma);





NUMA architecture

Benefits and limitations

Benefits:

- Ability to build system with big number of processors & cores;
  - Increase of performance (decrease of load on bus as a result of using “local” memory)
- Scalability;
- Ease of system partition on independent virtual machines

Limitations:

- Performance degradation when using non-local memory, or accessing the same shared memory from several nodes
- Design complexity as a result of maintaining cache coherency

Tested algorithms & system load

Block data processing algorithms with known interfaces:

- Compression
  - LZO, QLZ, Bcodec
- Encryption
  - AES (CBC, CTR mode)
- Hash
  - MD5

System load:

- Several types of stress load (CPU, TLB, QPI link)
- Customized via config (cpu #, intensity, etc.)

Parameters

Parameters of experiment are set via config file:

- Tested algorithm
- System load (type, intensity)
- Memory location (for NUMA architecture) for all buffers (in, out, stack)
- Data block size
- Input data (file or RND)

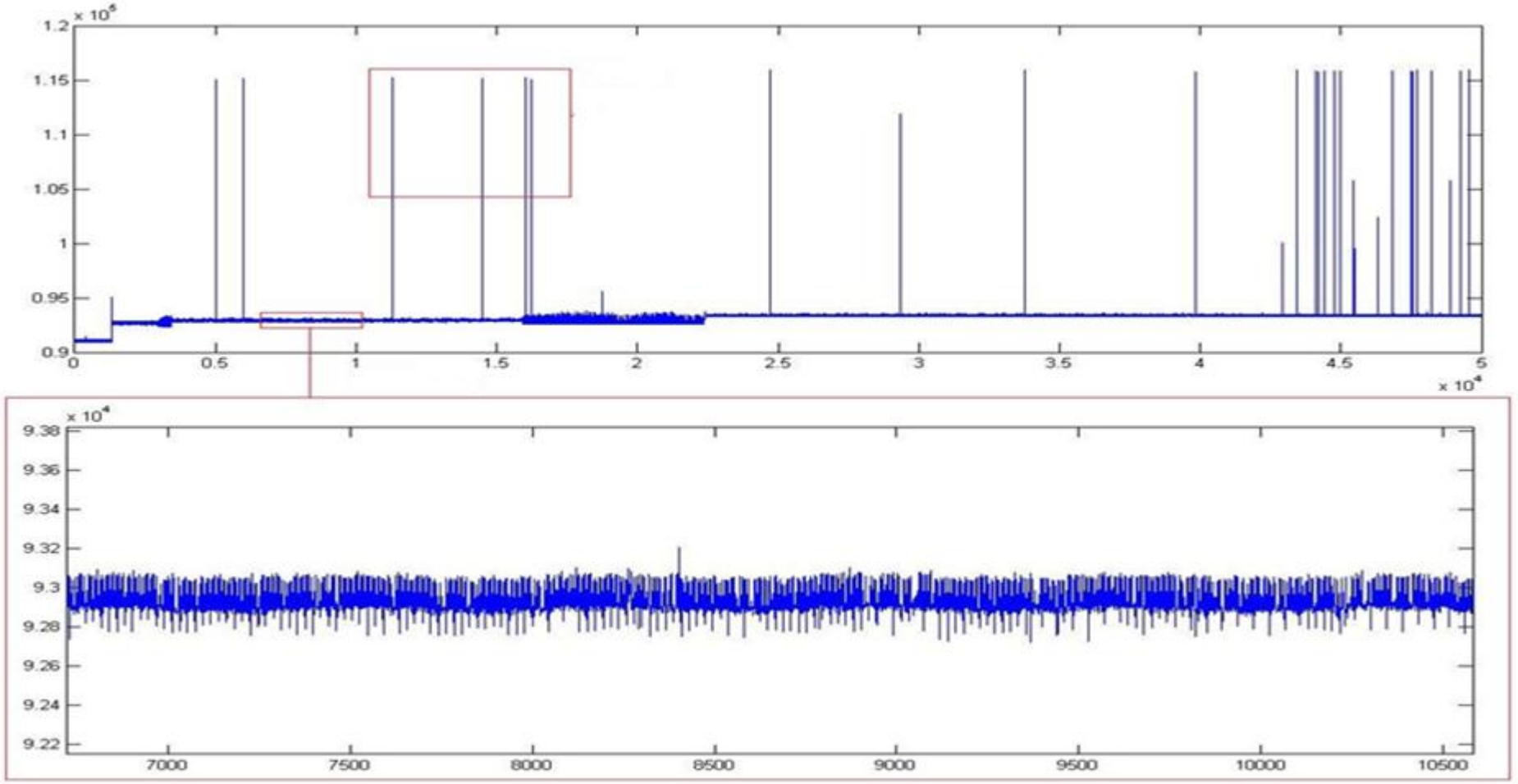
Additional preconfigured factors:

- Scheduler type
- Preemptible/Non-preemptible kernel

Measuring performance & statistics

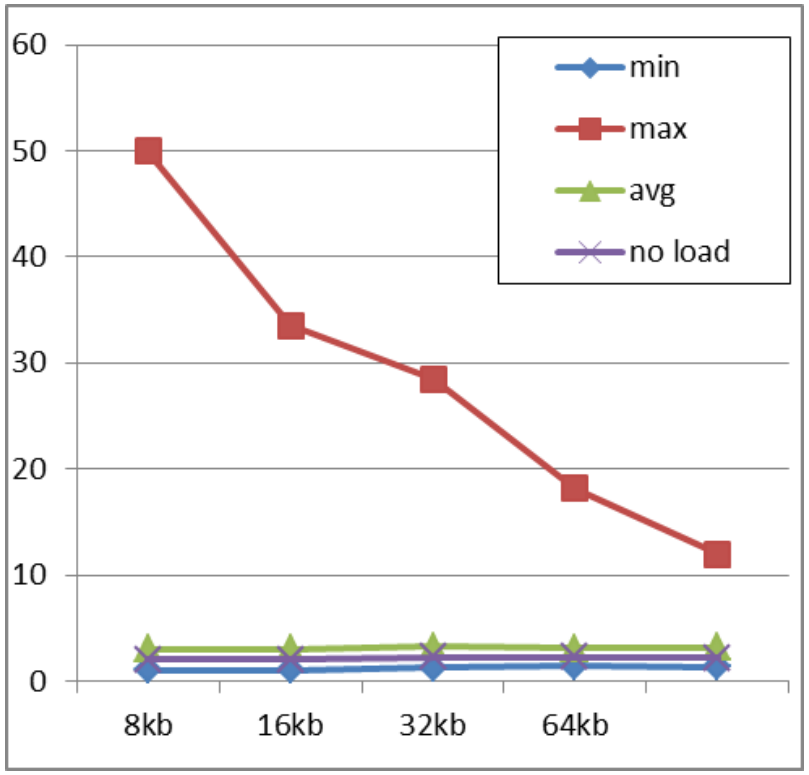
- Algorithms efficiency
  - in terms of execution time (average, min, max)
- Additional statistics can be measured using performance analysis tools like OProfile
  - Cash misses, TLB misses, CPU clock cycles, etc.

The experimental results: RAW data



Picture 1 - Statistics of the reference experiment

Results: BCodec



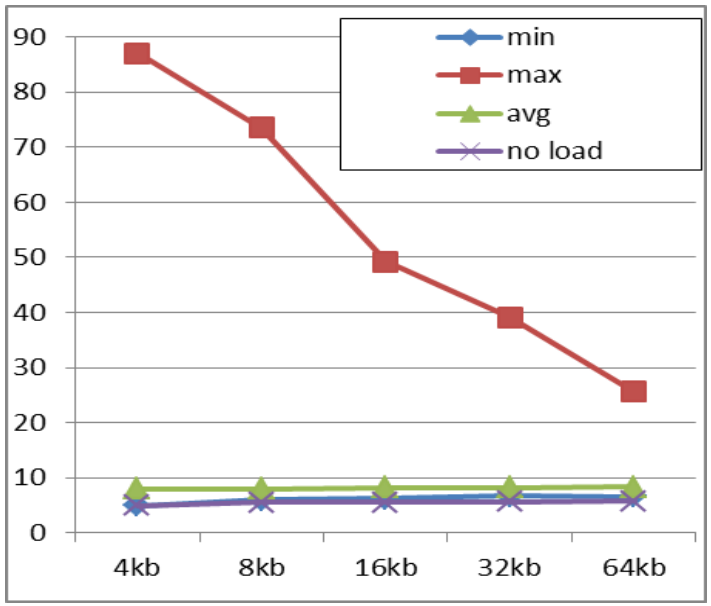
Max time (no load)

Max time (with load)

Average time

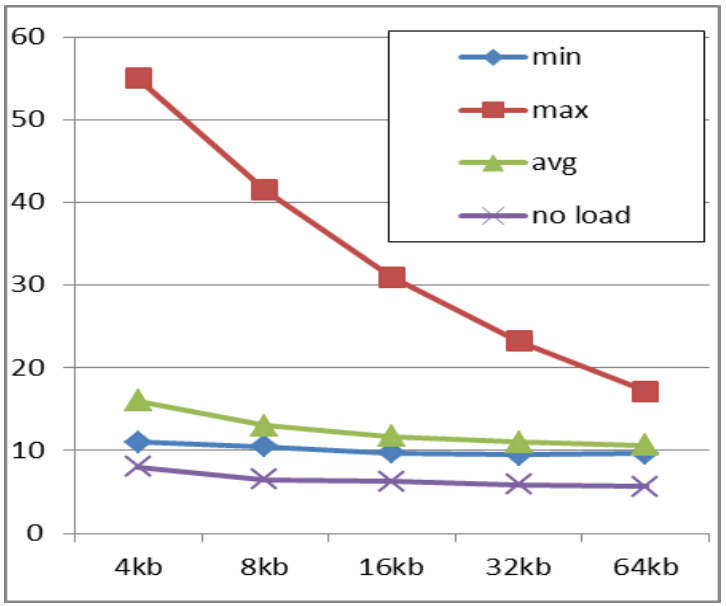
- Block data size does not influence algorithm performance (for average time)
- Under high load, bigger block size is preferable

Results: compression algorithms



LZO

VS



QuickLZ

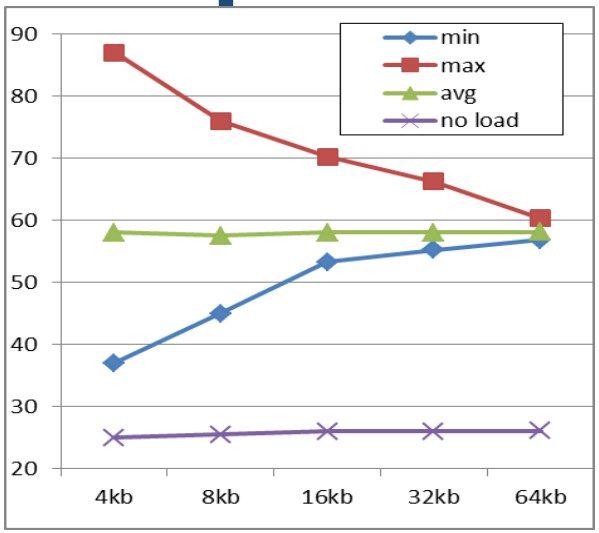
Max time (no load)

Max time (with load)

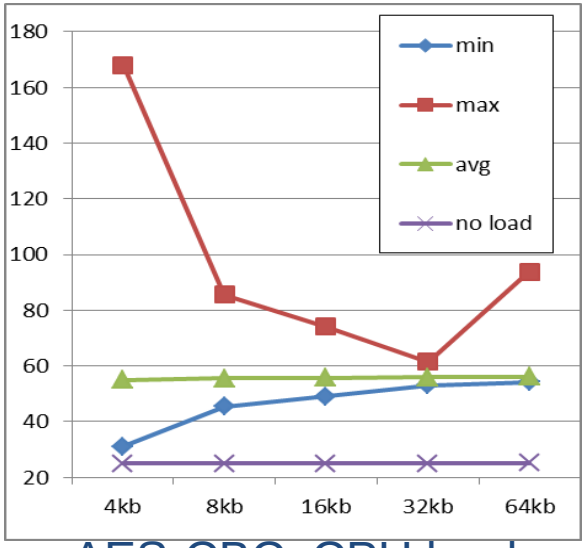
Average time

- Average time for LZO is better
- LZO is much worse for max time
- For block size 64K algorithms are comparable. Compression ratio should be taken into account

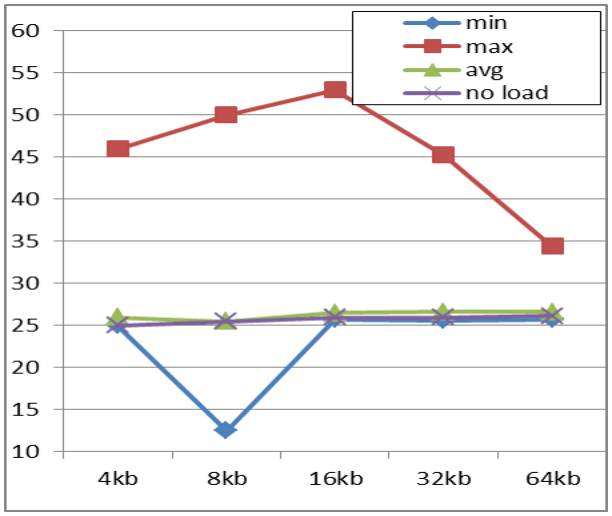
Comparison of the results



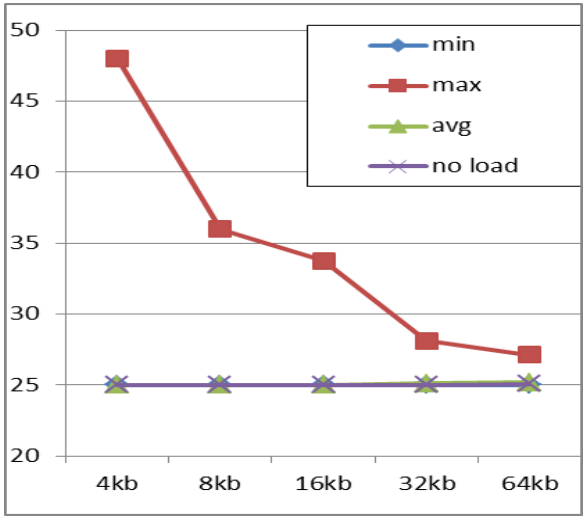
AES-CTR, CPU load



AES-CBC, CPU load



AES-CTR, QPI link load



AES-CBC, TLB load

Max time (no load)

Max time (with load)

Average time

Questions & Answers

Thank You!

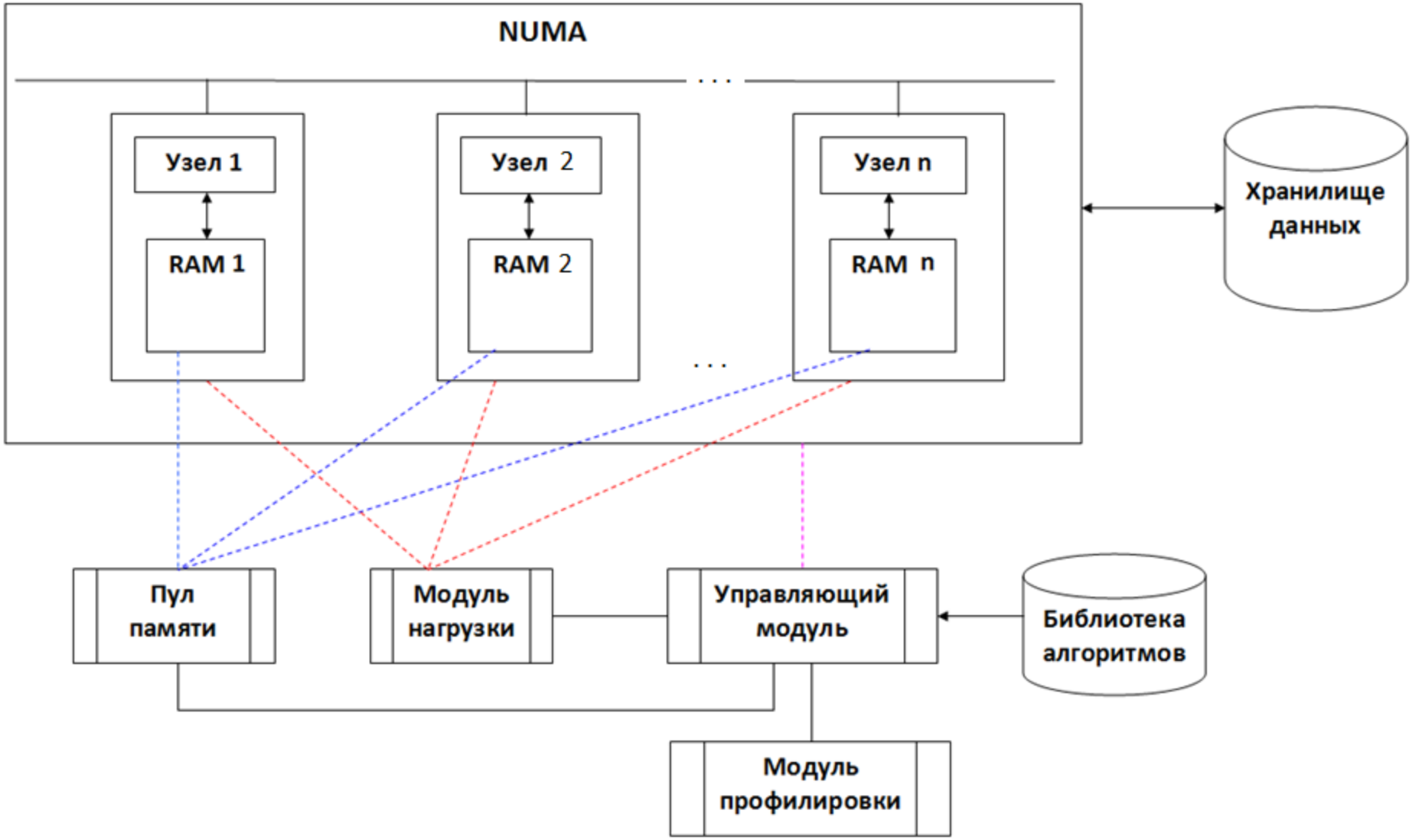
Future

- Carrying out experimentation in the XEN hypervisor environment
- More algorithms for testing
- Increasing the number of tested parameters
  - the number of time slices spent on a block of data, all data and etc.
- Experiments in the preemptive kernel mode
- Integration with other applications

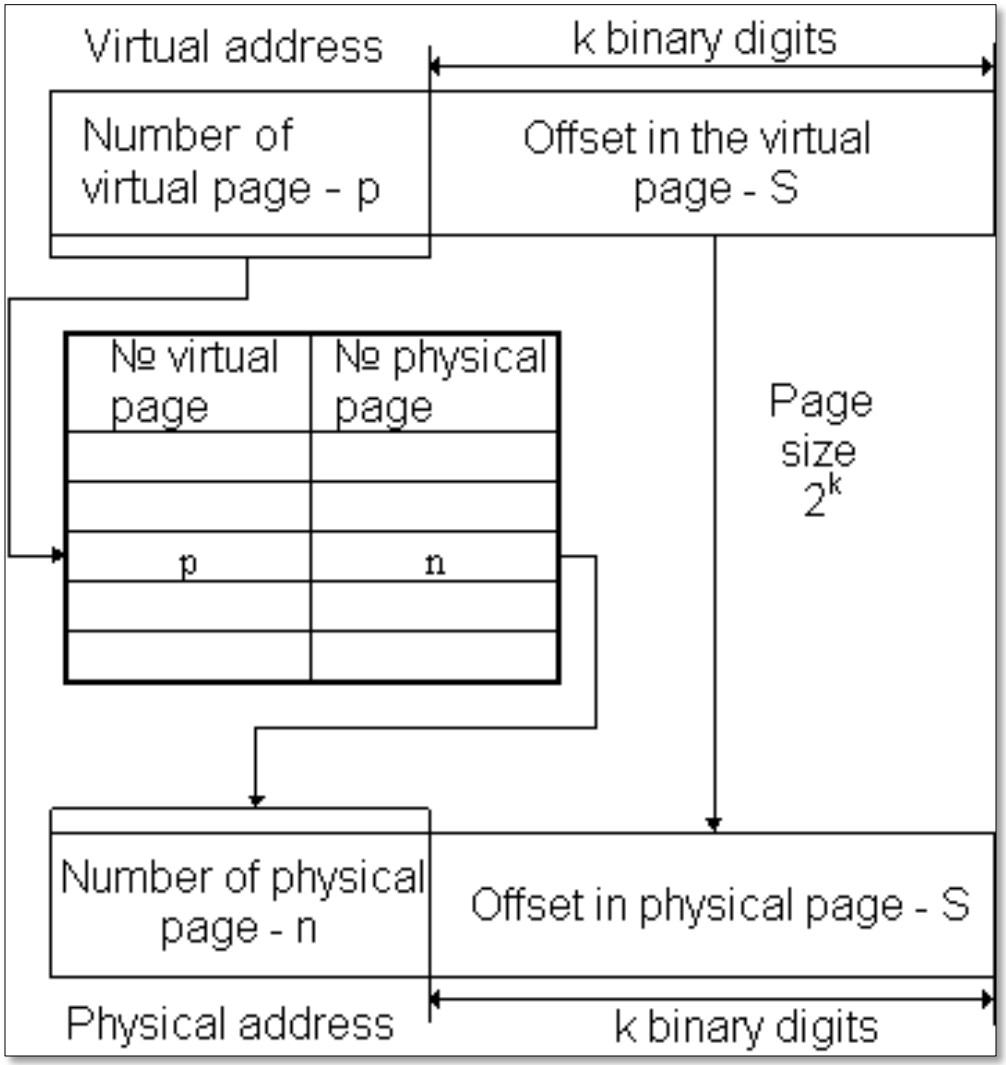
Tasks

- Investigation of Linux operating system kernel, detection of major factors that influence on algorithms performance
- Examination of interfaces for managing the kernel
- Creation of stand for empirical estimation of influence of revealed factors on algorithms performance

Архитектура стенда



Example of Ideas: Memory paging



Linux System memory is organized as pages of the volume of 4K.

If the memory is completely depleted, the OS will look for a long time unused memory pages to move them from memory to disk. If any of these pages is required, Linux restore them from disk.

Example of Ideas: Memory paging & Huge page

4 Kb – the size of standard memory pages.

2 Mb (x86) or 4 Mb (x86_64) – the size of huge page memory.

1) array 4 Mb = 4 Kb * 1024 pages

or 4 Mb = 2 Mb * 2 pages

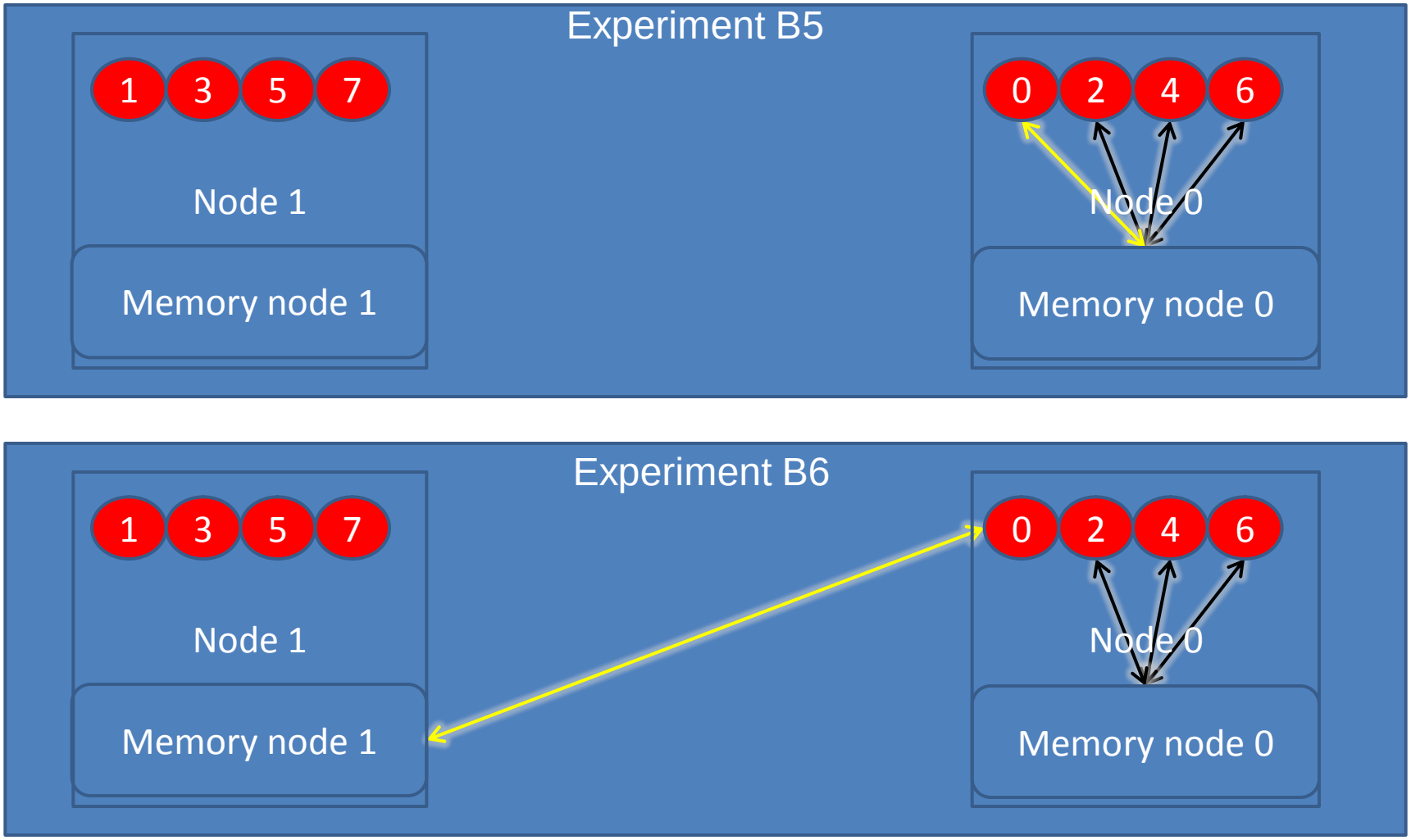
2) TLB (Translation Lookaside Buffer) – buffer that caches the last several transformations of memory addresses.

The advantages of using huge pages:

+ fewer pages required to allocate.

+ search in the TLB is faster (as TLB caches the few conversions)

Examples of configuration of the stand



Results of experiments with Algorithm 1

Reading from the local node	Reading from the non-local node
Experiment B1 - 8.76%	Experiment B2 - 18.42%
Experiment B3 - 18.91%	Experiment B4 - 40.59%
Experiment B5 - 11.1%	Experiment B6 - 9.88%
Experiment B7 - 28.1%	Experiment B8 - 21.4%
	Experiment B10 - 12.82%
Experiment B12 - 40.55%	Experiment B9 - 71.96%
	Experiment B11 - 4.18%

Results of experiments with MD5

Reading from the local node	Reading from the non-local node
Experiment B1 - 4.19%	Experiment B2 - 5.37%
Experiment B3 - 5.05%	Experiment B4 - 7.78%
Experiment B5 - 5.06 %	Experiment B6 - 4.99%
Experiment B7 - 4.96 %	Experiment B8 - 7.79%
	Experiment B10 - 5.22 %
Experiment B12 - 4.49 %	Experiment B9 - 6.63 %
	Experiment B11 - 4.12 %

Experience

- Now in the project involved 3 students and 1 PhD
- Obtained a lot of knowledge about the Linux kernel
- Experience of team work
- One student started working in a PowerPath