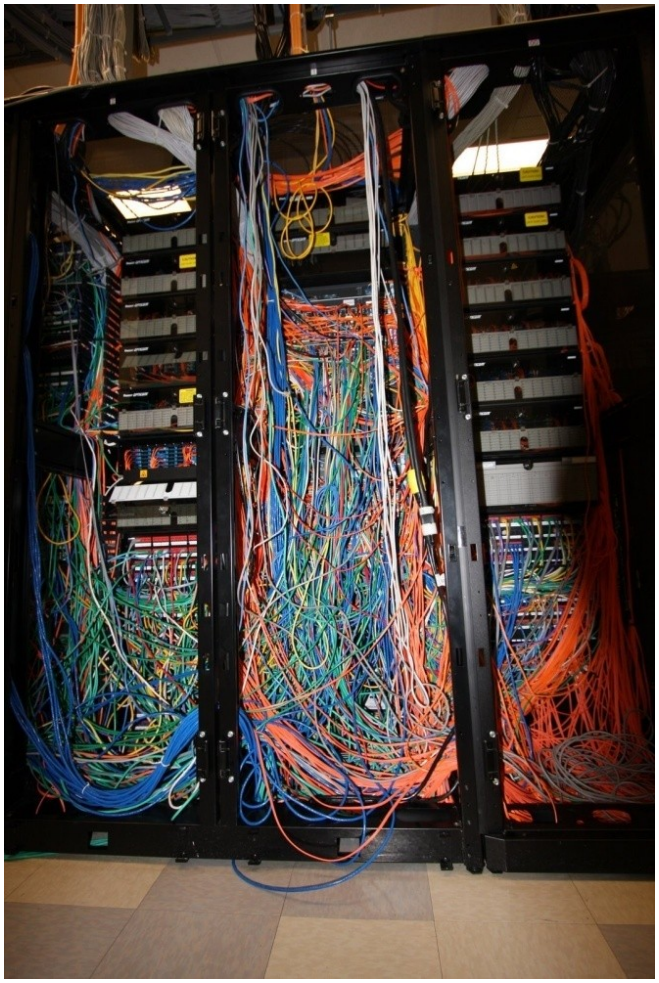
What if I push the red button?

Performance models for popular applications

Mastering complexity

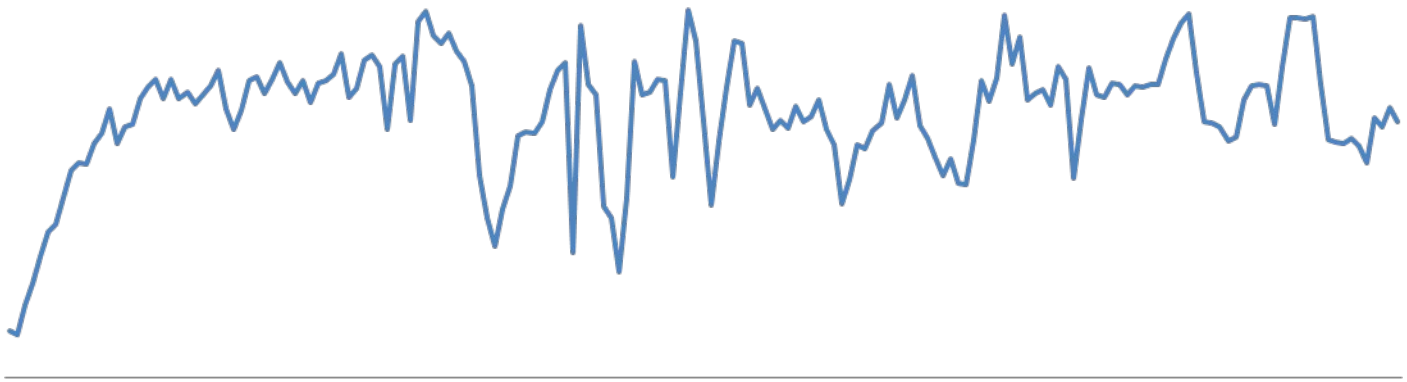
Issues in complex systems

- Security
- Dependability
- Manageability
- Scalability
- Performance



Performance Management

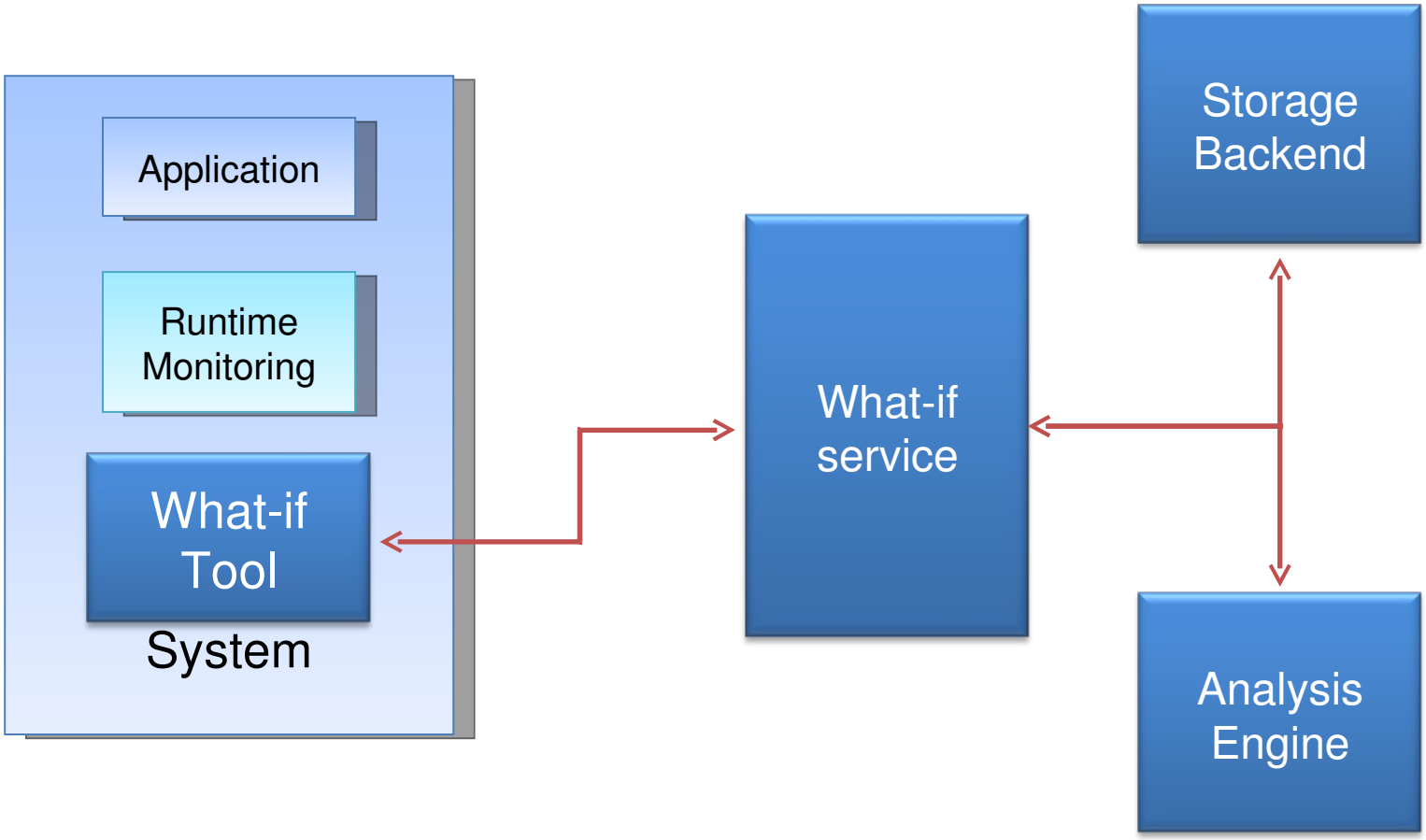
- Given a certain system configuration
  - Is my application performing normally?
  - Where does variance come from?
  - Should I invest money in faster disks or more RAM?



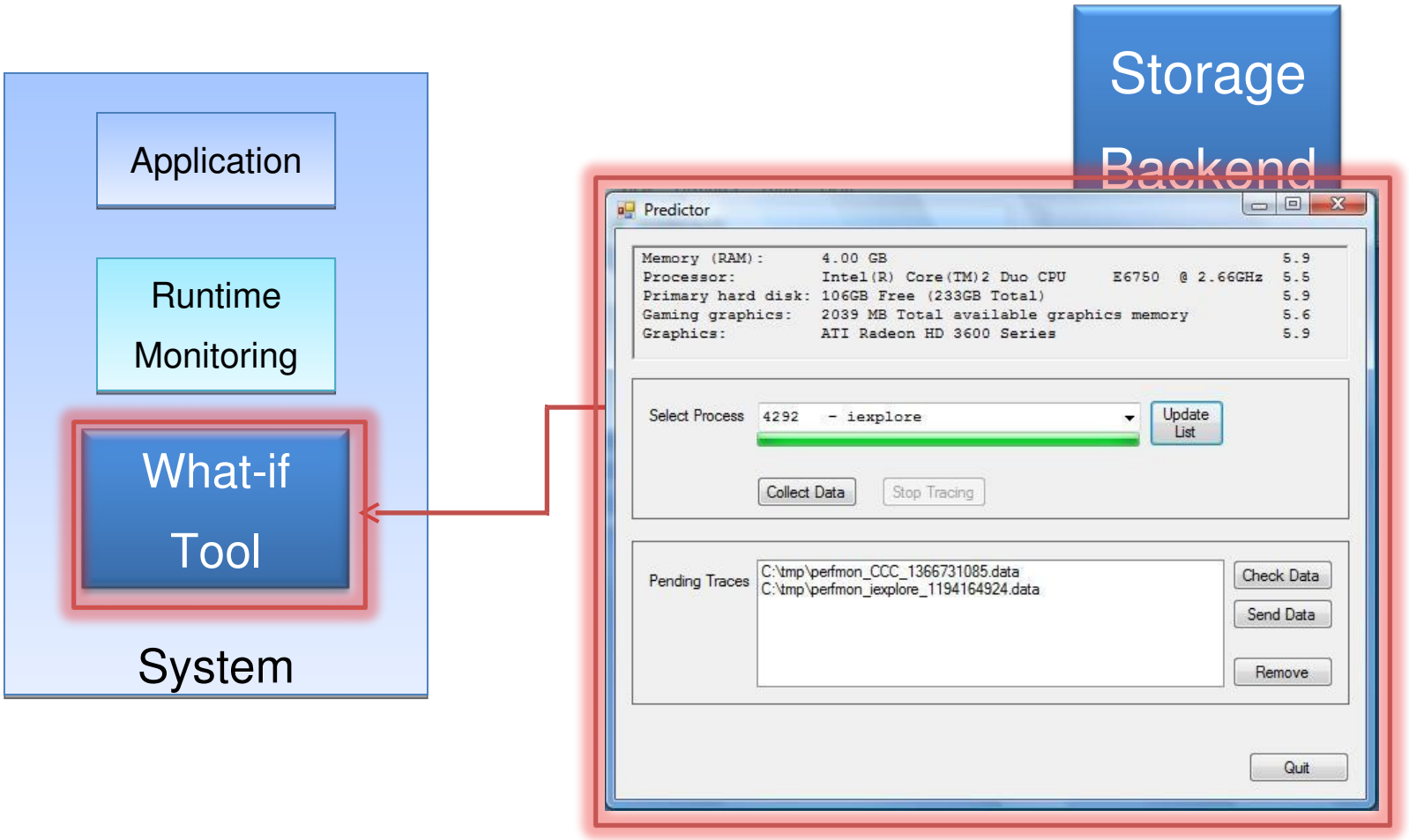
Our approach

- Focus on popular applications used by millions of users
- Runtime sampling
- Exploit statistical models for
  - Finding abnormal behaviour
  - Explaining variance in measurements
  - What-if analysis

Architecture Overview



Architecture Overview



Runtime Monitoring

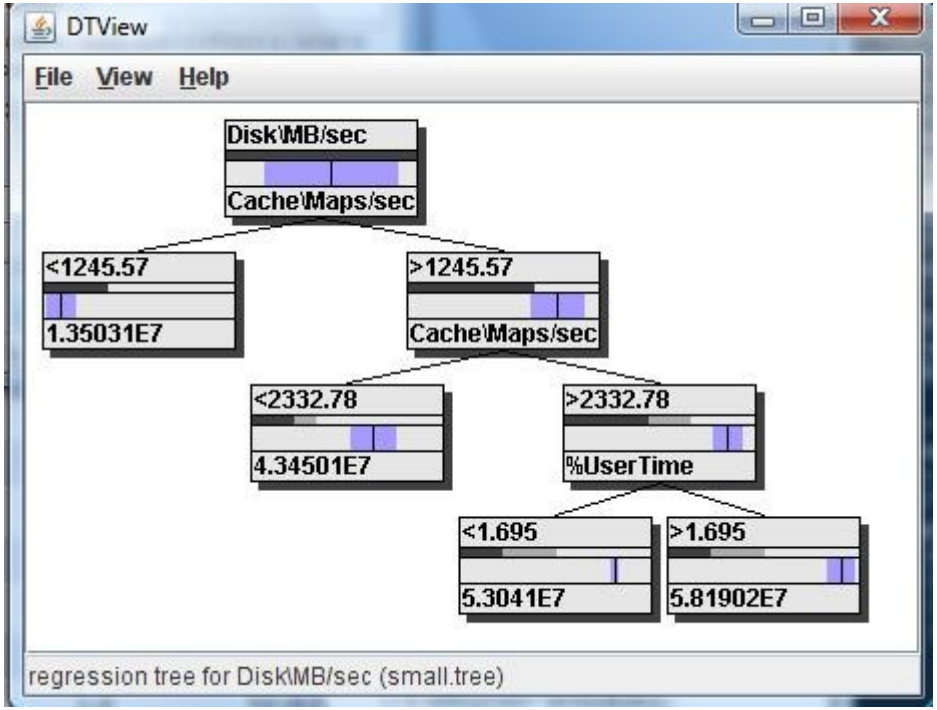
- Event Tracing for Windows
- Windows Performance Counters
- Focus on hardware resources

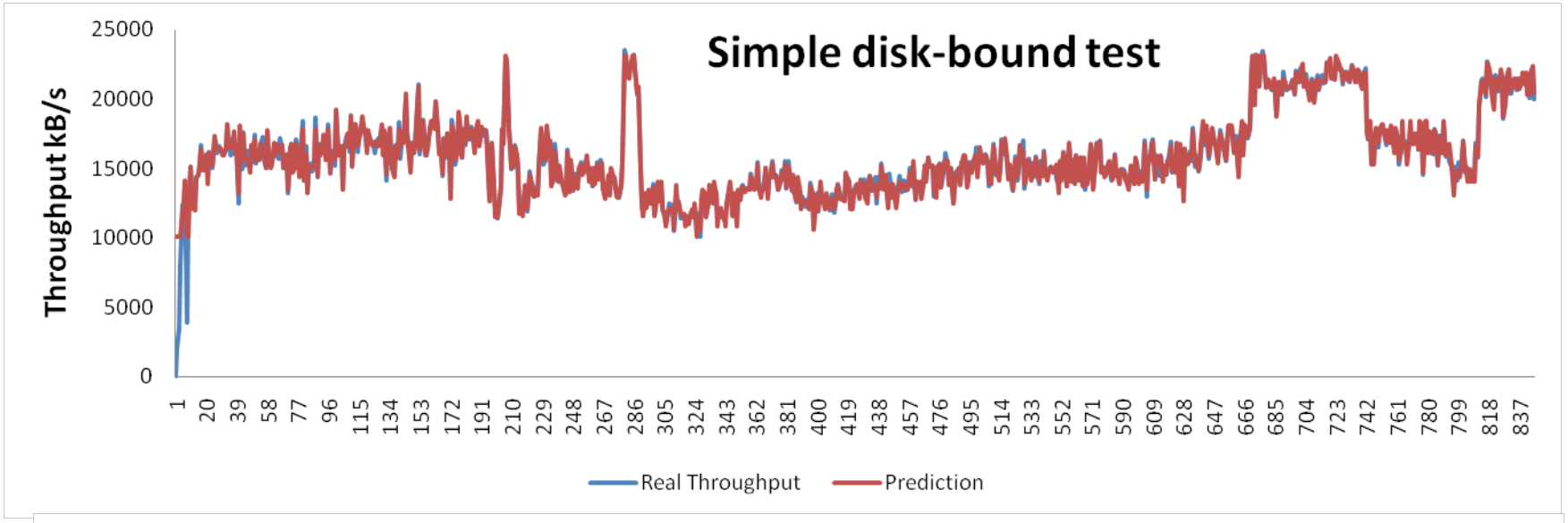
- **Does sampling data suffice to generate working models?**

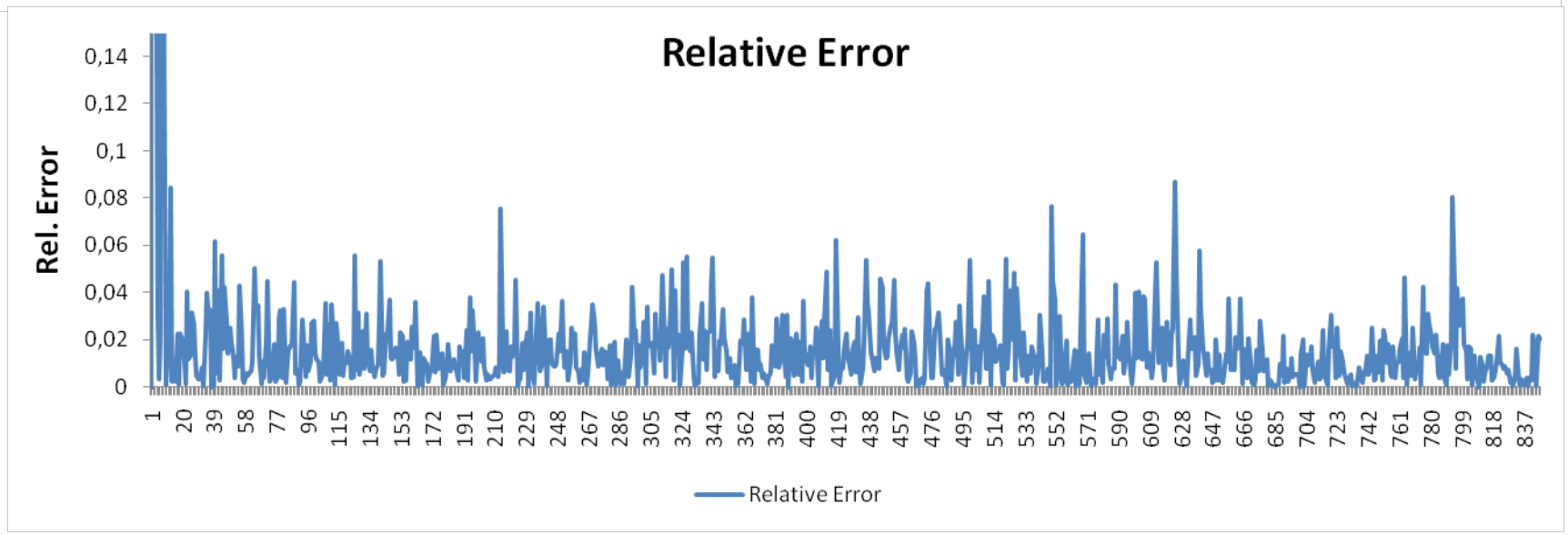
Cache\Maps/sec	%UserTime	Disk\MB/sec
387.67	1.07	7.93
700.49	0.38	15.55
779.99	0.47	17.02
1711.14	0.75	36.97
2359.01	1.65	52.17
2306.54	1.03	49.93
2818.32	5.94	62.02
2488.28	1.0	53.91
Cache\Maps/sec	%UserTime	Disk\MB/sec

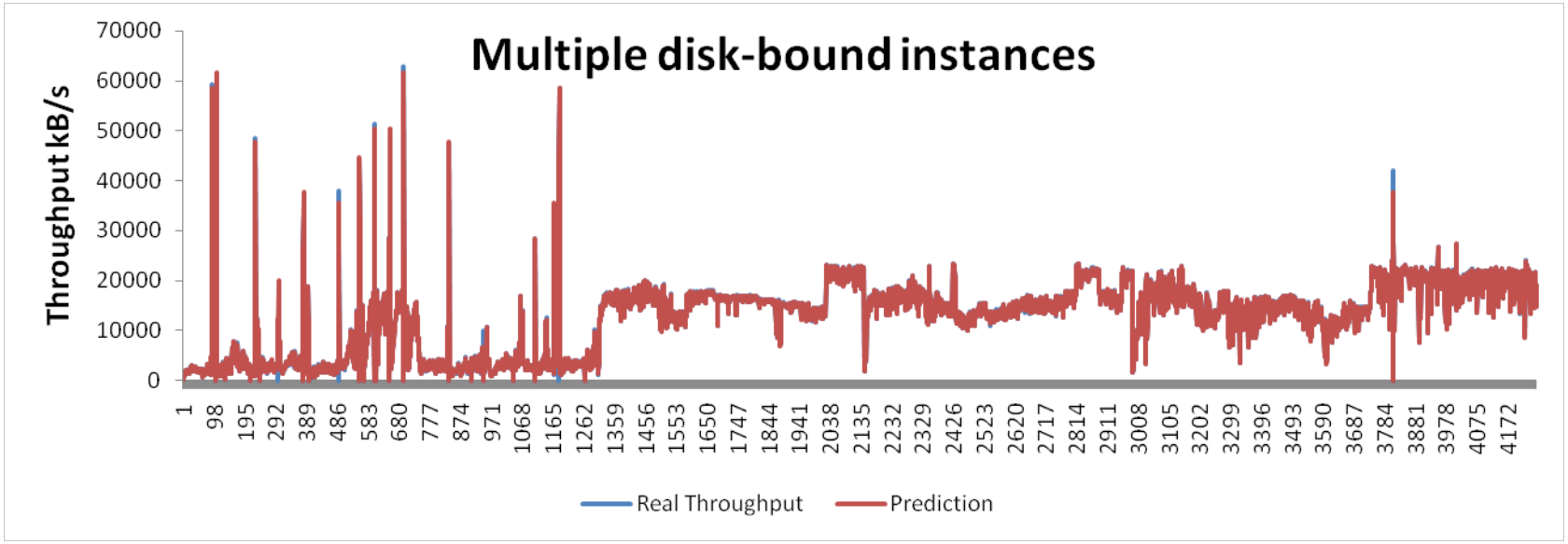
```
/*-----------------------------------------------------------------
  domains
-----------------------------------------------------------------*/
dom("Disk\\MB/sec") = IR [7.93109e+006,
6.20167e+007];
dom("Cache\\Maps/sec") = IR [387.67, 2818.32];
dom("%UserTime") = IR [0.38, 5.94];

/*-----------------------------------------------------------------
  regression tree
-----------------------------------------------------------------*/
dtree("Disk\\MB/sec") =
{ ("Cache\\Maps/sec"|1245.57)
  <:{ 1.35031e+007 ~3.98503e+006 [3] },
  >:{ ("Cache\\Maps/sec"|2332.78)
     <:{ 4.34501e+007 ~6.48238e+006 [2] },
     >:{ ("%UserTime"|1.695)
        <:{ 5.3041e+007 ~870106 [2] },
        >:{ 5.81902e+007 ~3.82654e+006 [2] }}}};

/*-----------------------------------------------------------------
  number of attributes: 3
  tree height        : 4
  number of nodes    : 7
  number of tuples   : 9
-----------------------------------------------------------------*
```

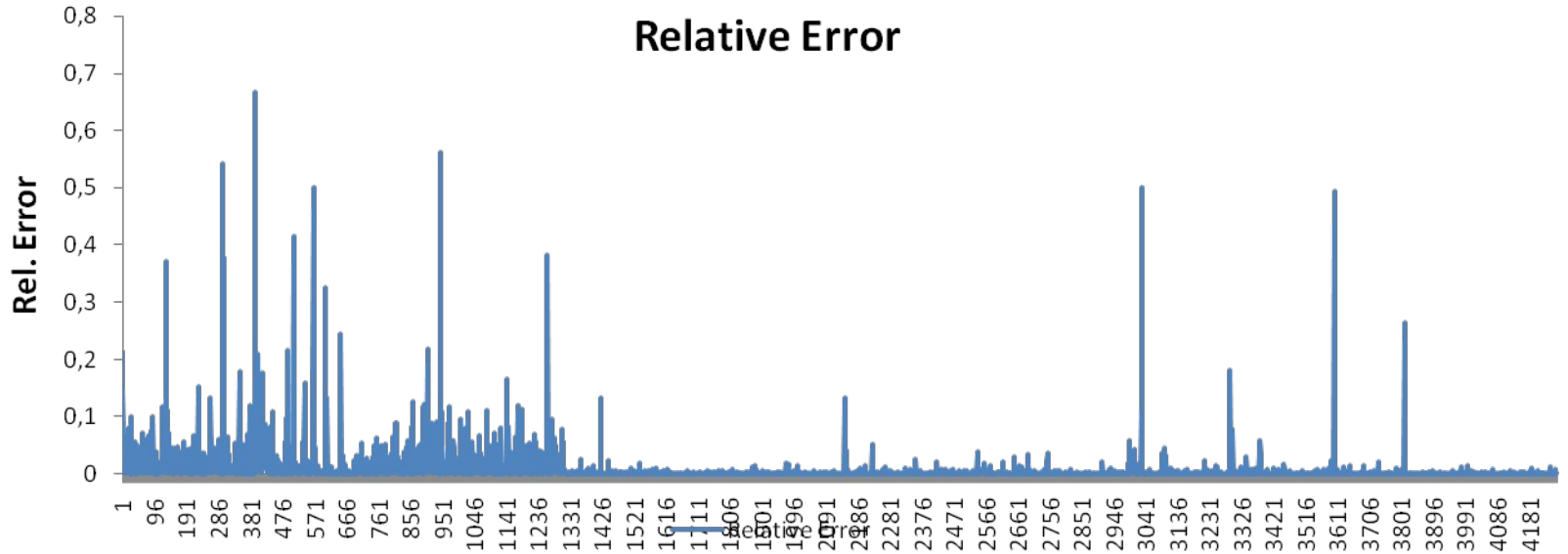
Modelling using Decision Trees

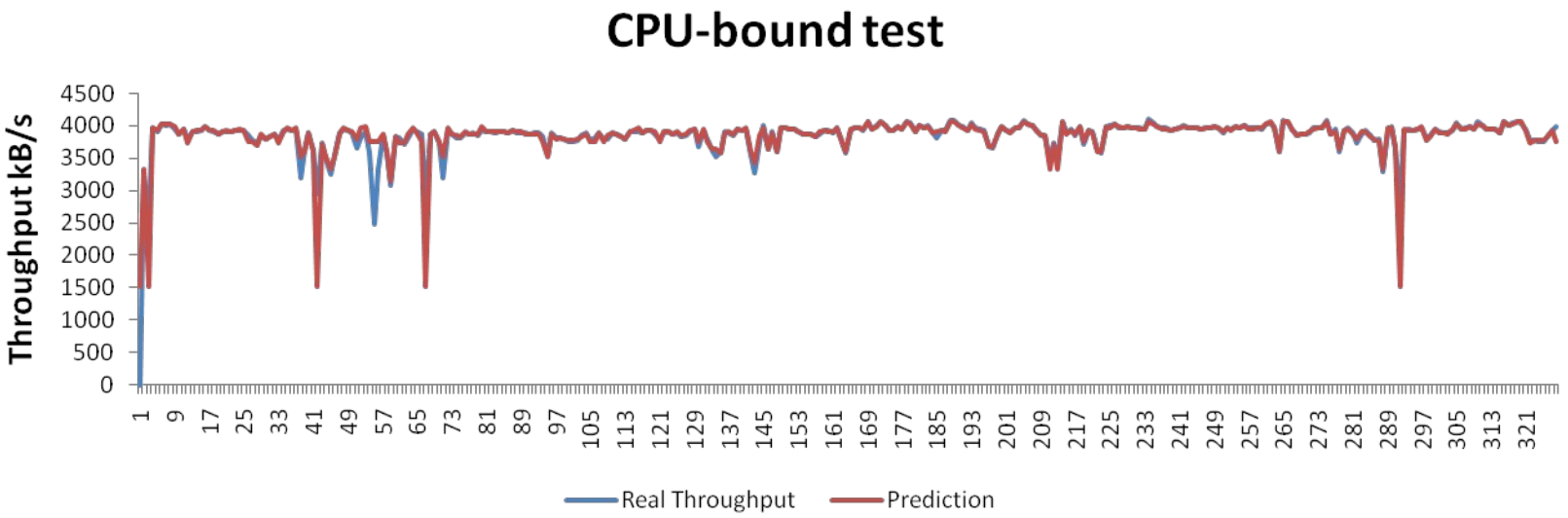


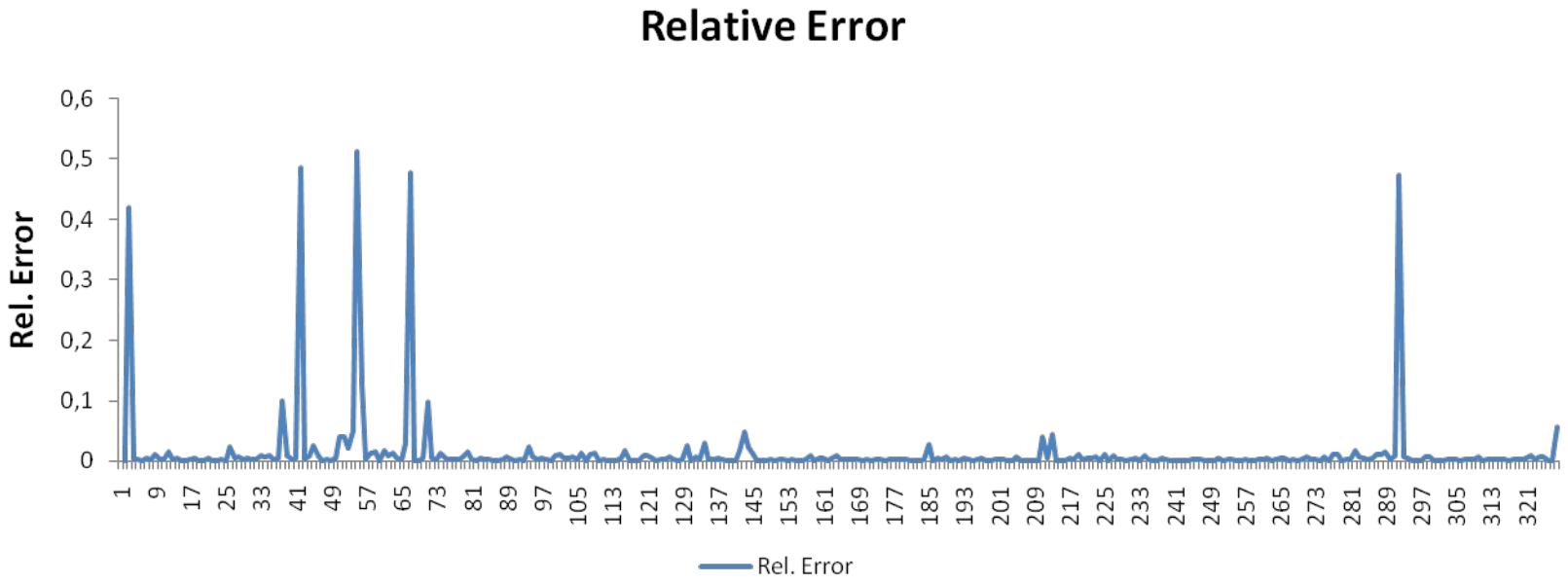

Simple disk-bound test
Throughput kB/s
25000
20000
15000
10000
5000
0
1 20 39 58 77 96 115 134 153 172 191 210 229 248 267 286 305 324 343 362 381 400 419 438 457 476 495 514 533 552 571 590 609 628 647 666 685 704 723 742 761 780 799 818 837
Real Throughput
Prediction

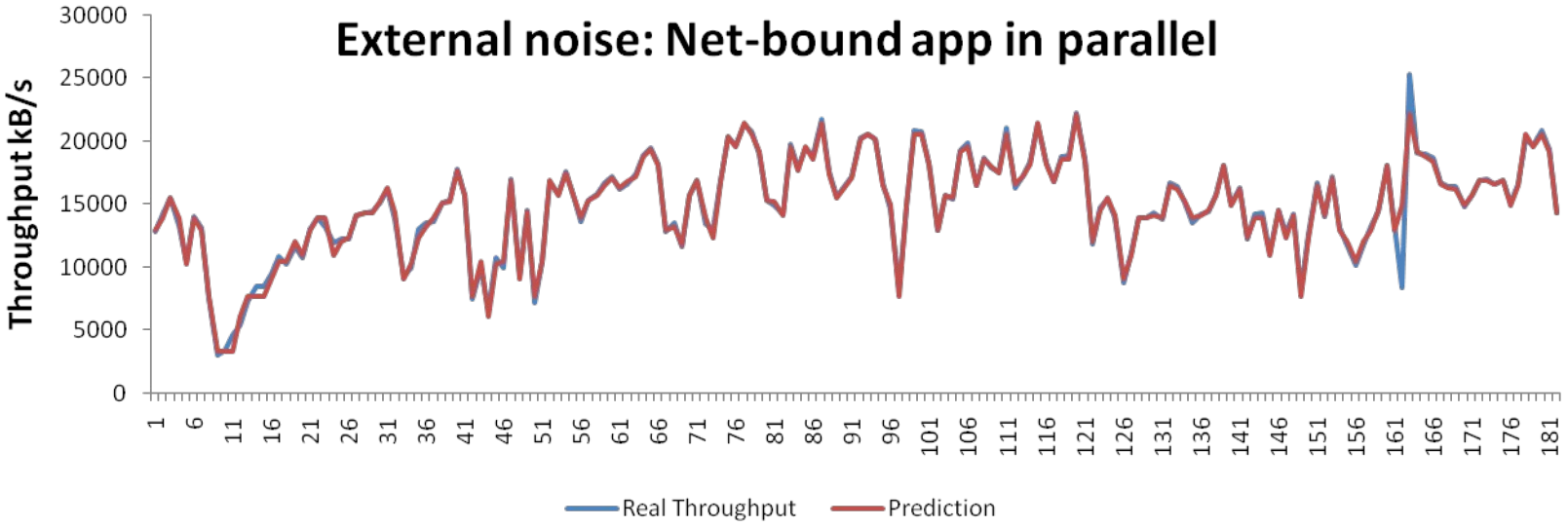

Relative Error
Rel. Error
0,14
0,12
0,1
0,08
0,06
0,04
0,02
0
1 20 39 58 77 96 115 134 153 172 191 210 229 248 267 286 305 324 343 362 381 400 419 438 457 476 495 514 533 552 571 590 609 628 647 666 685 704 723 742 761 780 799 818 837
Relative Error

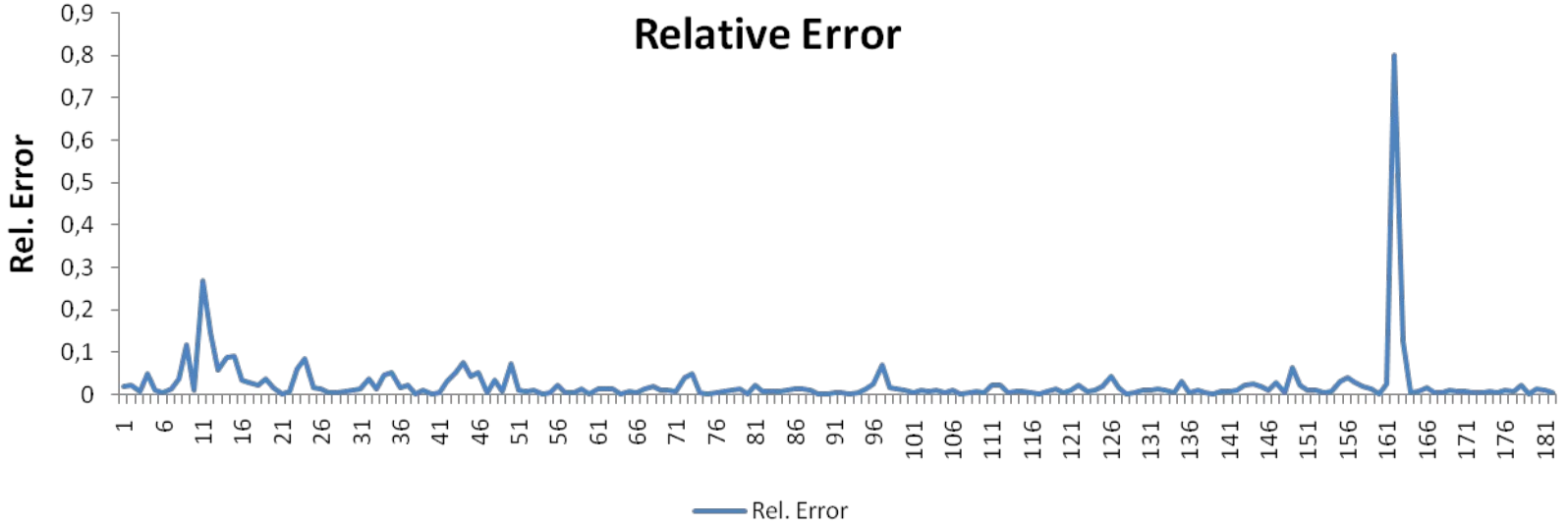

Multiple disk-bound instances
Throughput kB/s
70000
60000
50000
40000
30000
20000
10000
0
1
98
195
292
389
486
583
680
777
874
971
1068
1165
1262
1359
1456
1553
1650
1747
1844
1941
2038
2135
2232
2329
2426
2523
2620
2717
2814
2911
3008
3105
3202
3299
3396
3493
3590
3687
3784
3881
3978
4075
4172
Real Throughput
Prediction


Relative Error
Rel. Error
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
1
96
191
286
381
476
571
666
761
856
951
1046
1141
1236
1331
1426
1521
1616
1711
1806
1901
1996
2091
2186
2281
2376
2471
2566
2661
2756
2851
2946
3041
3136
3231
3326
3421
3516
3611
3706
3801
3896
3991
4086
4181
Relative Error


CPU-bound test
Throughput kB/s
4500
4000
3500
3000
2500
2000
1500
1000
500
0
1 9 17 25 33 41 49 57 65 73 81 89 97 105 113 121 129 137 145 153 161 169 177 185 193 201 209 217 225 233 241 249 257 265 273 281 289 297 305 313 321
Real Throughput
Prediction


Relative Error
Rel. Error
0,6
0,5
0,4
0,3
0,2
0,1
0
1 9 17 25 33 41 49 57 65 73 81 89 97 105 113 121 129 137 145 153 161 169 177 185 193 201 209 217 225 233 241 249 257 265 273 281 289 297 305 313 321
Rel. Error

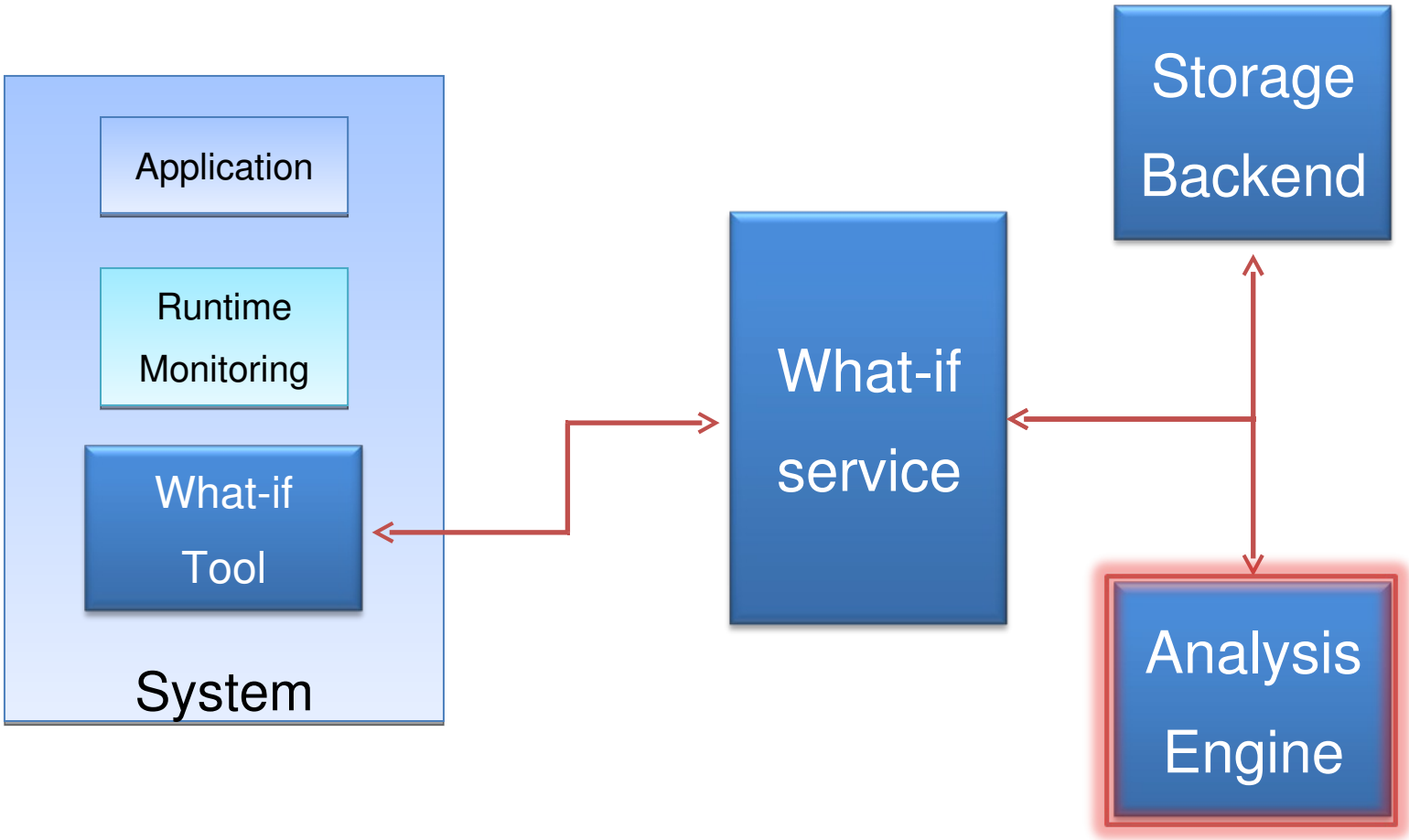

External noise: Net-bound app in parallel
Throughput kB/s
30000
25000
20000
15000
10000
5000
0
1 6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96 101 106 111 116 121 126 131 136 141 146 151 156 161 166 171 176 181
Real Throughput
Prediction


Relative Error
Rel. Error
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
1 6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96 101 106 111 116 121 126 131 136 141 146 151 156 161 166 171 176 181
Rel. Error

Results

- Sampling-based predictions work.
  - Fast predictions
  - Reasonably fast tree generation

Architecture Overview



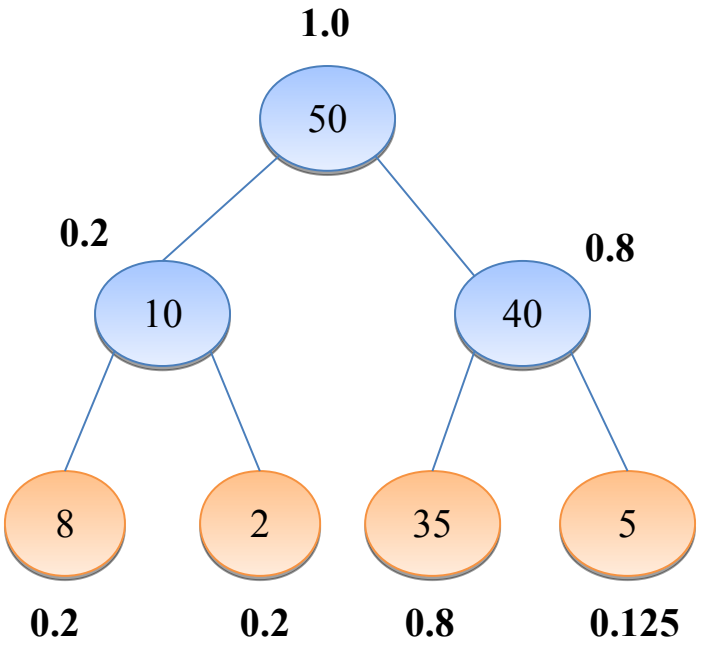
Where does variance come from?

- Insufficient local resources
- Resource consumption by other (unrelated) programs
- Long-term performance degradation
- Dependence on external resources

- **Can we exploit performance models to help us with these problems?**

Normal behavior?

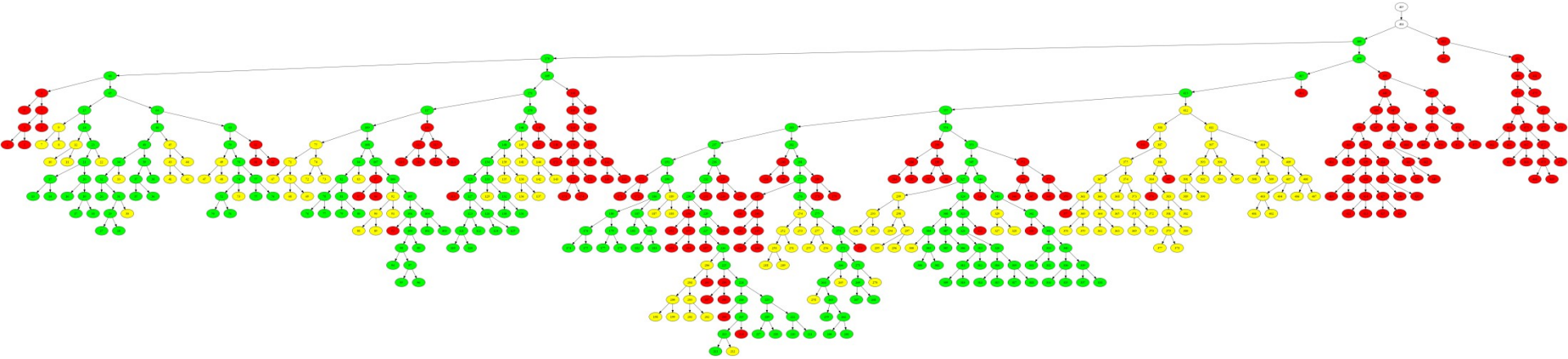
- Our scenario: Collect performance data from millions of users
- Resulting decision tree contains everything
  - “Good” vs. “bad” values?
  - Assumption: applications work good for most of the users
  - Add weight component to each branch of the tree



$$w(root) = 1.0$$

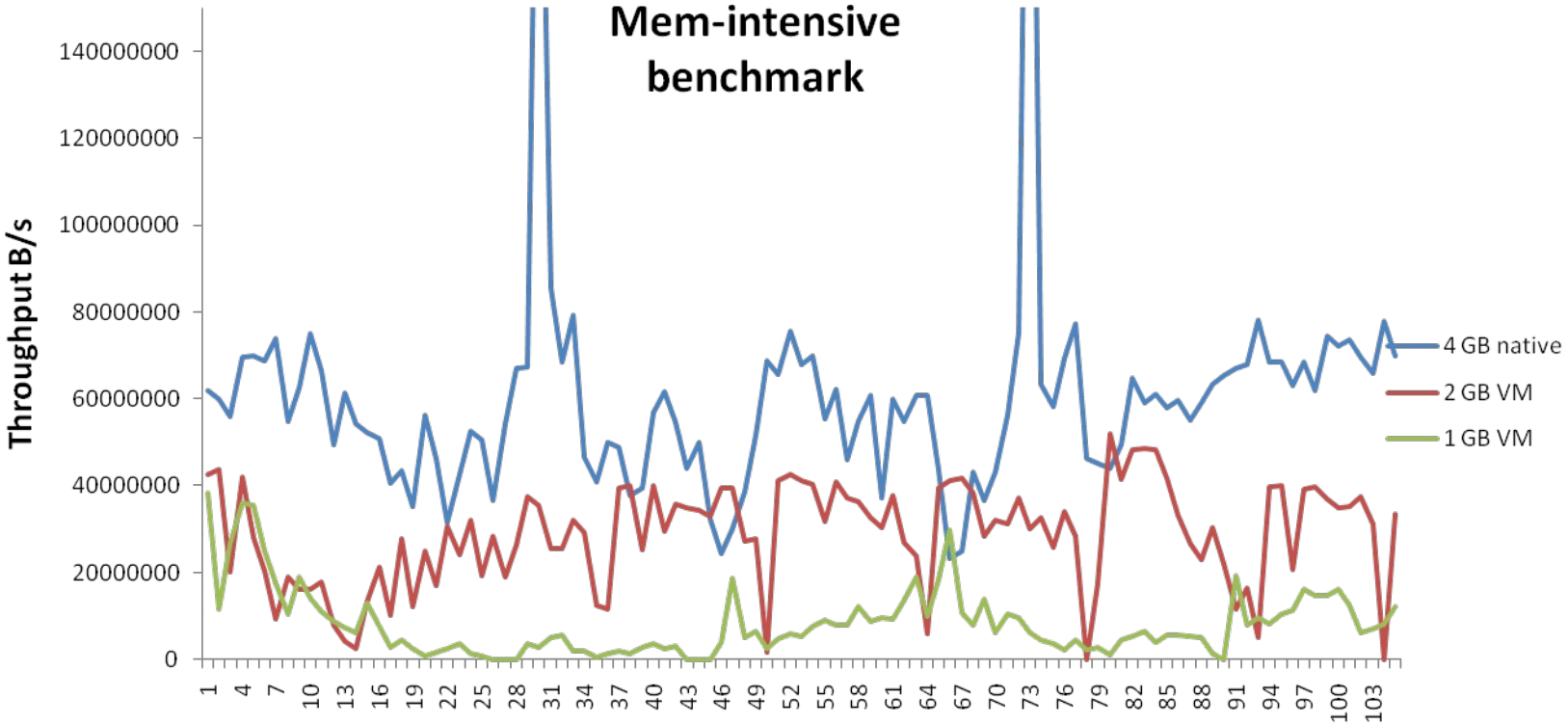
$$w(node) = \min\left(w(parent), \frac{datasets(node)}{datasets(parent)}\right)$$

Normal behavior?



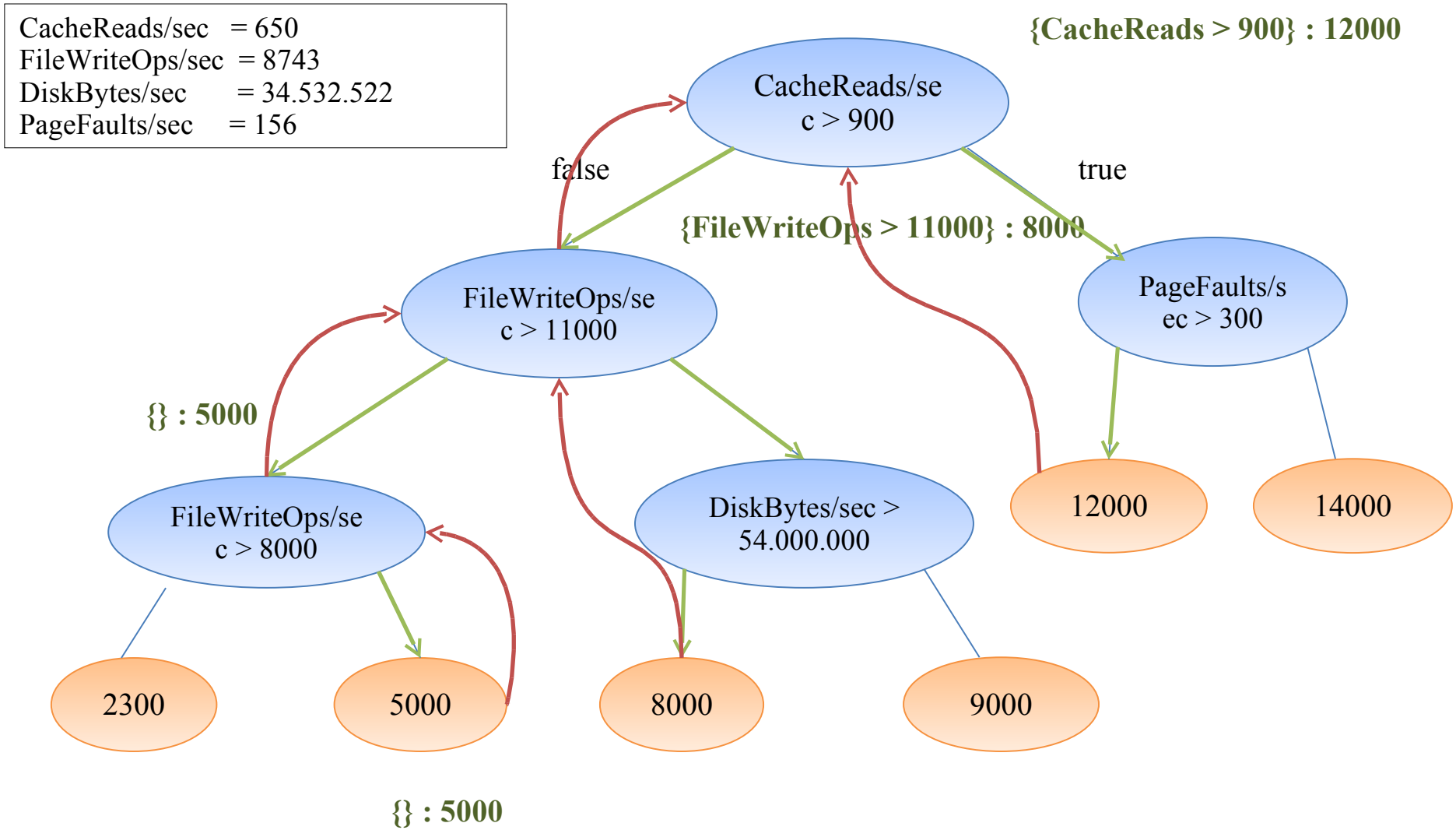
What if I push the red button?

Correlation vs. Causality



How can I improve performance?

(simplified tree for Process\DiskIOBytes/sec)



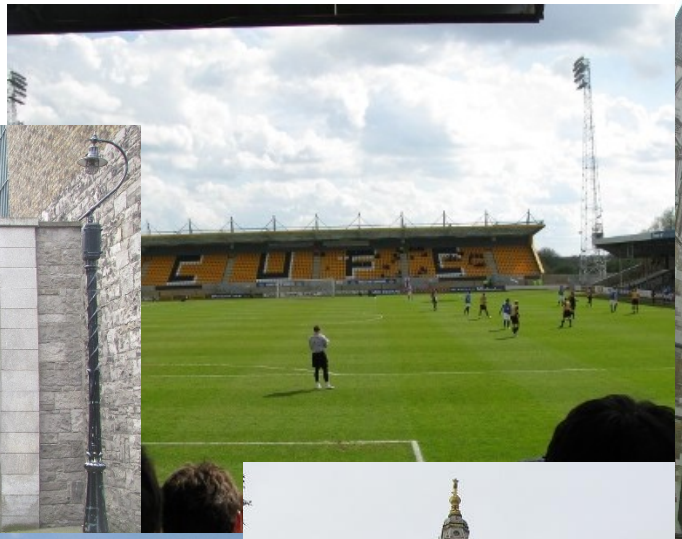
And how do I change CacheReads/sec?

- We do not necessarily know
  - Whether changing a resource changes the attribute.
  - Which resource should be changed.
- Next steps:
  - Cluster configurations based on their correlation to measured attributes (similar to [Cohen05])
  - Nearest-Neighbour Search for best improvement with minimal configuration change

Coming soon

- Study a real application
  - Find out root causes for variance
  - Correlate configuration to attributes
- Trace refinement loop
- Larger User study
  - Collect data from many users
  - Compare users' observations – point out problems
  - Make suggestions for solving problems


GUINNESS
ST. JAMES'S GATE BREWERY, DUBLIN



Related work

- [Aguilera03] M. Aguilera et al. "*Performance Debugging for Distributed Systems of Black Boxes*", SOSP 2003
- [Borg08] http://www.borgelt.net/dtree.html
- [Cohen05] I. Cohen et al. "*Capturing, Indexing, Clustering, and Retrieving System History*", SOSP 2005
- [Perl93] S. Perl, W. Weihl "*Performance Assertion Checking*", SIGOPS 1993
- [Reynolds06] P. Reynolds et al. "*Pip: Detecting the Unexpected in Distributed Systems*", NSDI 2006
- [Shen05] Kai Shen et al. "*I/O System Performance Debugging using Model-driven Anomaly Characterization*", FAST 2005
- [Stewart05] C. Stewart, K. Shen "*Performance Modelling and System Management for Multi-Component Online Services*", NSDI 2005
- [Thereska08] E. Thereska, Gregory R. Granger "*IRONModell: Robust Performance Models in the Wild*", SIGMETRICS 2008
- [Wang04] H. Wang et al. *Automatic Misconfiguration Troubleshooting with PeerPressure, OSDI2004*