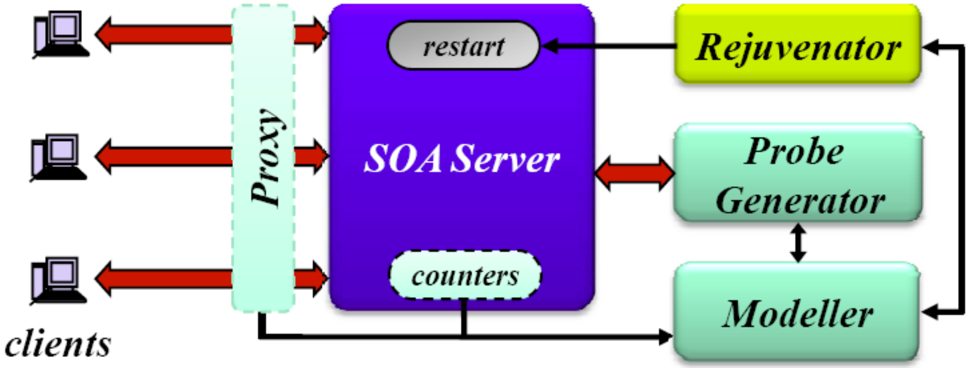
Using Machine Learning for Non-Intrusive Modeling and Prediction of Software Aging



Artur Andrzejak

Zuse-Institute Berlin (ZIB)
Berlin, Germany
andrzejak[at]zib.de

work with
Luis Silva, Univ. of Coimbra

What is Software Aging?

- "progressive degradation of the running software that may lead to system crashes or undesirable hang ups"
- caused by memory leaks, unterminated threads, unreleased file descriptors, accumulated numerical errors…
- a more suitable name: software *state* aging or software *running image* aging

Examples of Aging Applications

- This problem has been reported in *operating systems*, *web-servers*, *enterprise clusters*, *OLTP systems*, *spacecraft systems*, *grid middleware*...
  - What is the uptime of your MS Windows? ☺

> ***"Software aging was responsible for the loss of the Patriot anti-missile in the Gulf-war. The solution for the problem was to reboot and restart the Patriot Software every 8 hours".***
>
> ***Science, page 1347, March 13, 1992***

- Especially vulnerable are always-on / long-running applications
  - web services, enterprise applications
    - e.g. serious aging in Apache Axis 1.3, 1.4

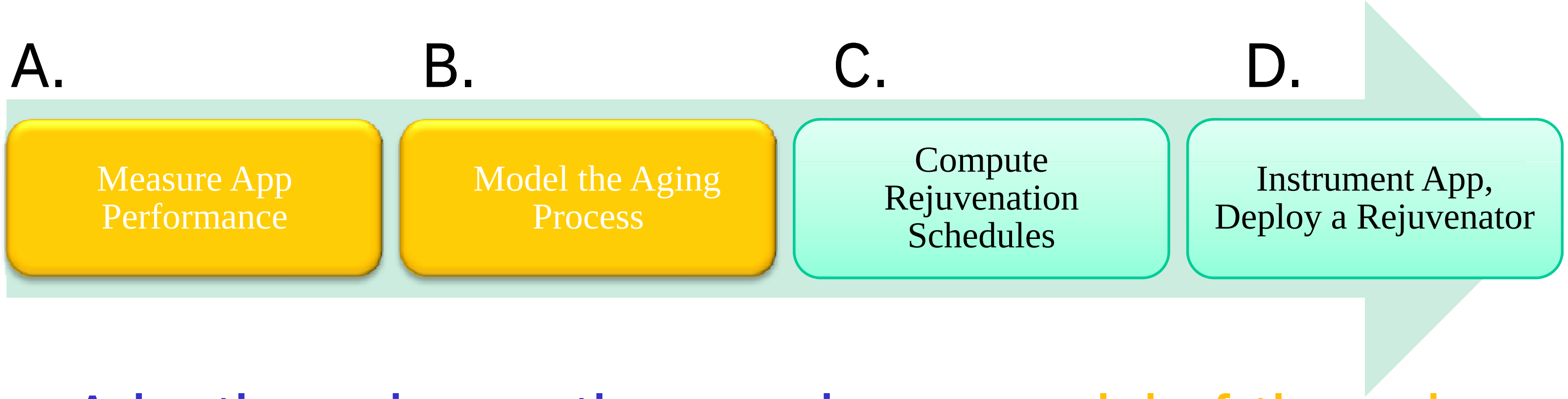
How to overcome aging?

- Ideal solution: *fix the errors!* BUT..
- In practical terms, most software deployed in enterprises are black box applications
  - commercial / 3rd party: lack of source code
  - large or legacy: too costly / complex to debug
- In such cases the only practical solution is rejuvenation
  - restart of an application, VM, OS, cluster
  - accepted as a de-facto "anti-aging potion" of the IT management

Adaptive Rejuvenation

- Rejuvenation has essential costs and penalties
  - causes availability interruption
  - reduces the average app performance
- Therefore rejuvenate in a "smart" way, such as:
  - only if performance drops below a critical level
  - according to some optimization criteria, e.g. maximizing the average performance
- This is called adaptive rejuvenation
  - considered as more efficient
  - area of this paper

Four Steps of Implementing An Adaptive Rejuvenation



A. Measure App Performance

B. Model the Aging Process

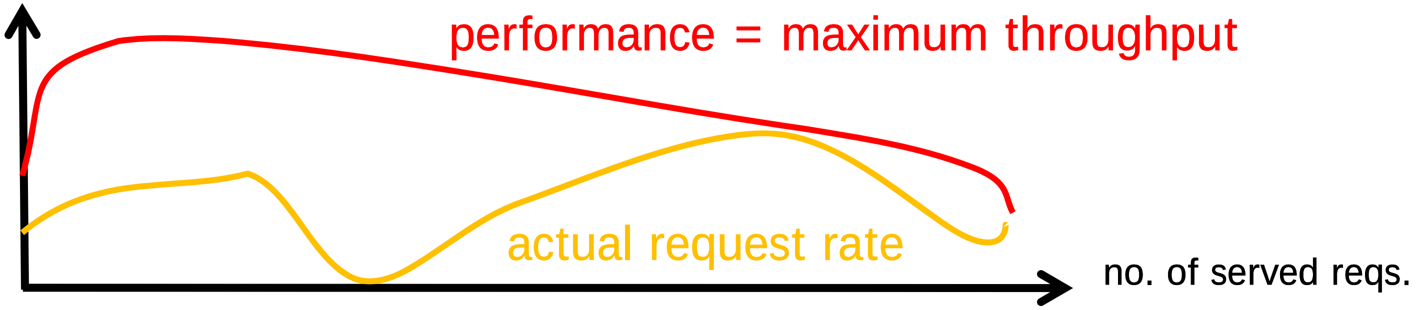
C. Compute Rejuvenation Schedules

D. Instrument App, Deploy a Rejuvenator

- Adaptive rejuvenation requires a model of the aging process
  - essential e.g. to decide whether it is not too late or too early to rejuvenate
  - usually *equivalent to modeling/predicting the degradation of app performance*
- This work focuses on measuring and modeling of performance degradation - phases A & B

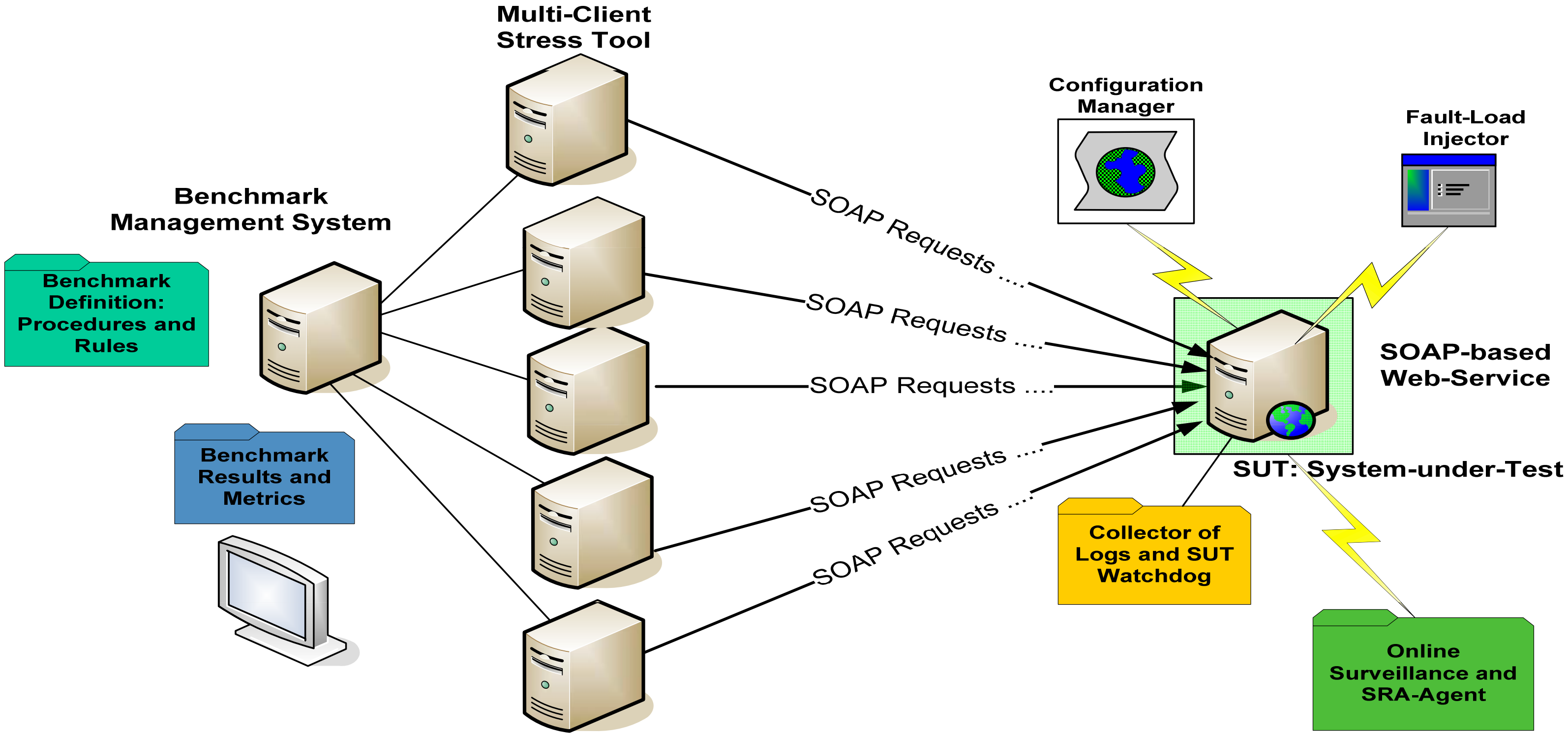
Assumptions

- We consider "server-type" apps
  - e.g. web or application servers
  - stateless or have short session time (compared to aging speed)
  - all experiments: TPC-W benchmark in Java instrumented with a memory leak injector
- Our proxy for the aging progress is ..
  - application performance = *maximum* number of requests which could be served per second
    - i.e. maximum throughput
    - actual request rate might be much lower



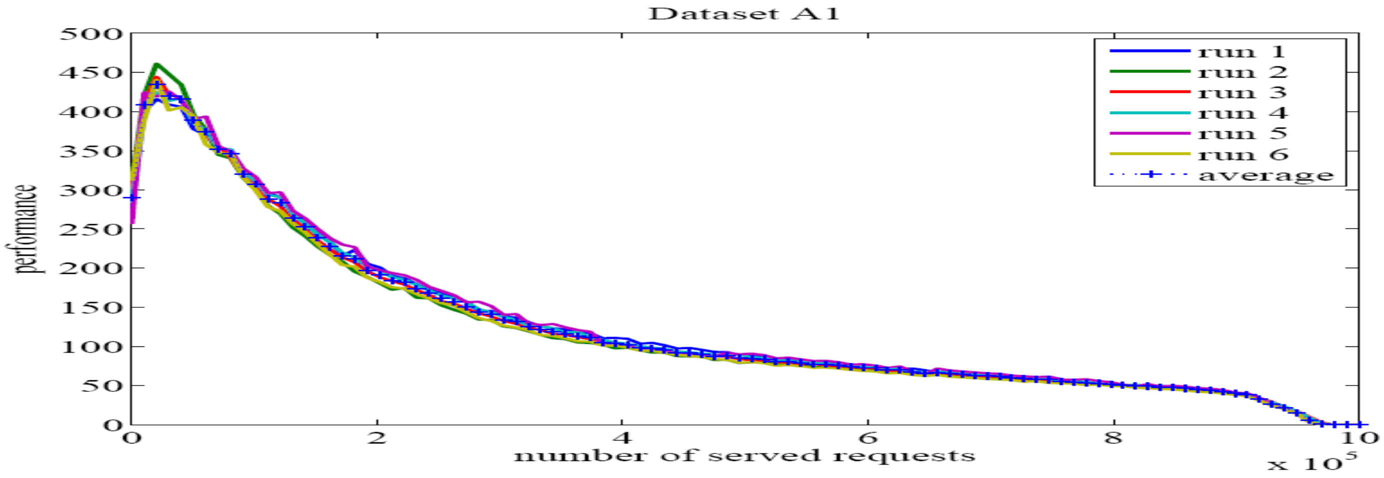
Measuring under Burst Distribution

- Measuring performance degradation under laboratory conditions:
  - put the application under full load & count the number of serviced requests per second



Measuring under Burst Distribution

- How does this look like?
  - Curves for six experiments (same parameters)



- Drawbacks:
  - Only laboratory scenario: high setup effort, little realistic
  - Other types of request rates are not covered

How to Measure in a Production Environment?

- Problem: *you can only measure performance if the maximum service rate is attained*
  - rarely the case under real conditions
- Solution:
  - Interweave real requests and high-request-rate probes used to measure the performance

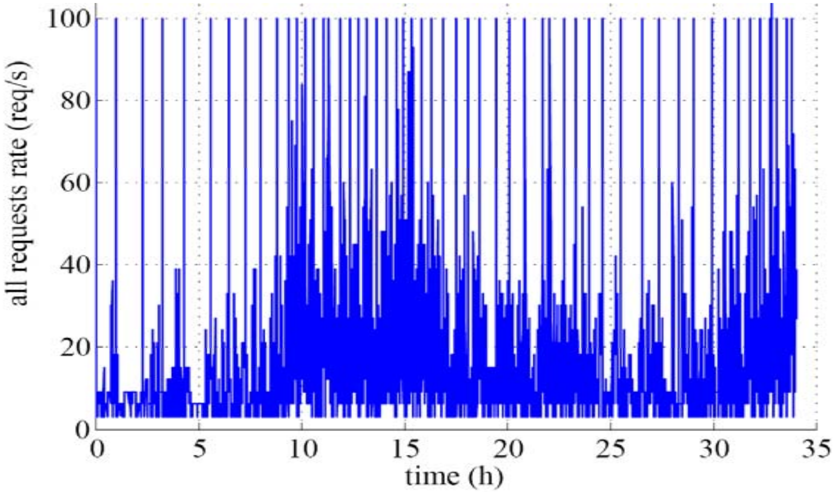
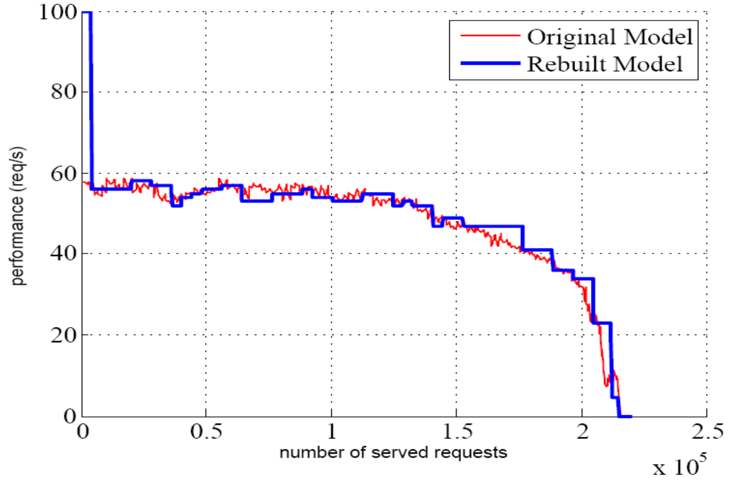


Figure:

- real request rates of the campus-wide web server at the University of Saskatchewan (see M. Arlitt / C.L. Williamson 1996)
- interweaved are request probes with rate of 100 req/sec

Measuring – Accuracy, Efficiency



Accuracy (optical comparison, MSE)

- Example: TPC-W benchmark, app. server aging due injected memory leaks
- red: aging process found under "laboratory conditions"
- blue: the probing approach

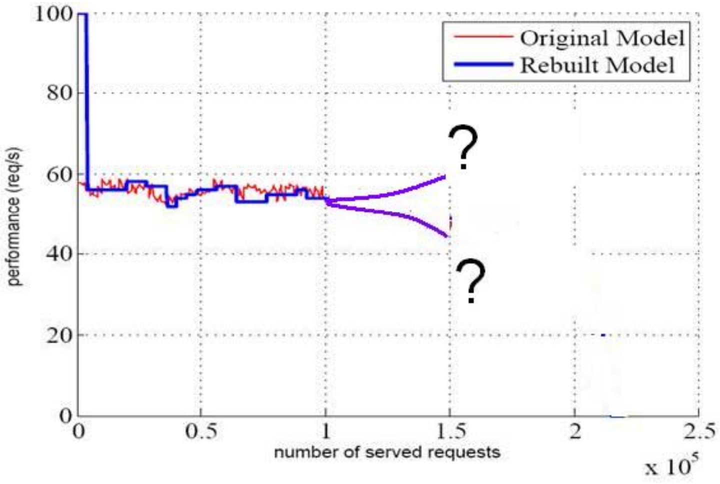
Efficiency

- No free lunch: probing accelerates aging
- How much?
- With accuracy as above, probing requests are about 1.2%
- (MSE =9.32, MAE = 0.60)

With some instrumentation the method can be used in a production environment

Modeling Aging Processes

- In order to schedule rejuvenation, raise alerts, compute average performance..
- .. we need an "oracle" which tells us app performance as a function of..
  - number of served requests (our case)
  - some other suitable variables, e.g. memory usage, time,..



- We have been working on two methods:
  - previously: spline-based models for stable or simple aging processes
  - this work: machine learning-based models for more complex or less deterministic processes

Models of Time Series

- Essentially, we need to predict a time series (= performance)
- There are many generic and proprietary methods
  - traditional: moving averages, AR, ARIMA, ..
  - voting & boosting schemes: Network Weather Service
  - periodicity and pattern mining schemes
  - ...
- However, we have used classifiers known from machine learning / data mining

What are Classifiers?

- Essentially it is a function which *learns* its output value from examples
- function inputs are called attributes, in our study:
  - transformed data about current performance and number of served requests
- output is an element from some fixed set, in our study:
  - a discretized performance level

	Attribute$_1$	…	Attribute$_n$	**Output**
Example 1	0.2	…	0.7	[0.1-0.2]
Example k	0.4	…	1.5	[0.7-0.8]
Prediction	0.3	…	0.9	**?**

} learn

← predict

Why Classifiers?

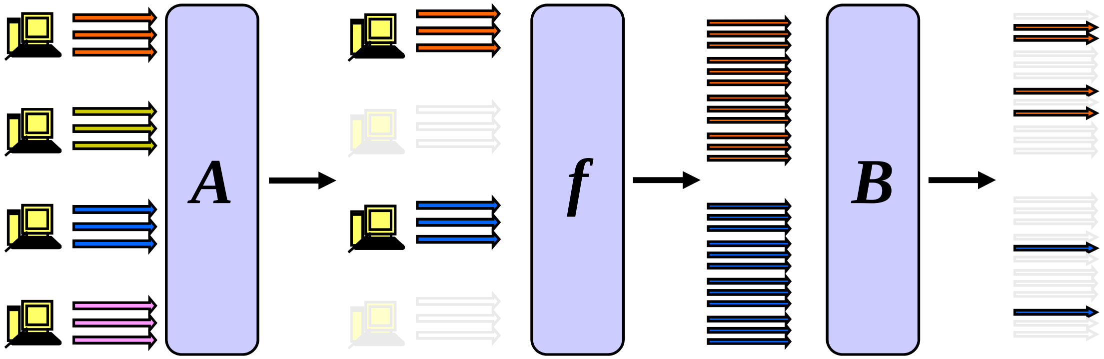
1. There is an abundance of sophisticated and tuned state-of-the-art algorithms, including:
    - decision trees
    - Bayesian Methods
    - Support Vector Machines – the "Holy Grail" of last years
    - neural nets
        - just a special case, and NOT the best one
2. There are good & fast libraries (Java, Matlab, C/C++)
3. Most important:
    - classifiers allow for multiple inputs which can be either continuous or discrete

Why Multiple Inputs?

- Theory and practice shows that no classification / prediction algorithm is significantly better than the others
  - "*No free lunch theorem*" of data mining (Wolpert)
  - Also results of this paper: 3 "serious" alg. are equally good
  - Why?
    1. if patterns exist, they are quite obvious
    2. you cannot make gold out of s...
- So to get better models / predictions we can only
  - use more inputs
  - preprocess data better
  - give "hints" by incorporating prior knowledge
    - most "new prediction methods" are in fact doing this
- We have focused on the first two items

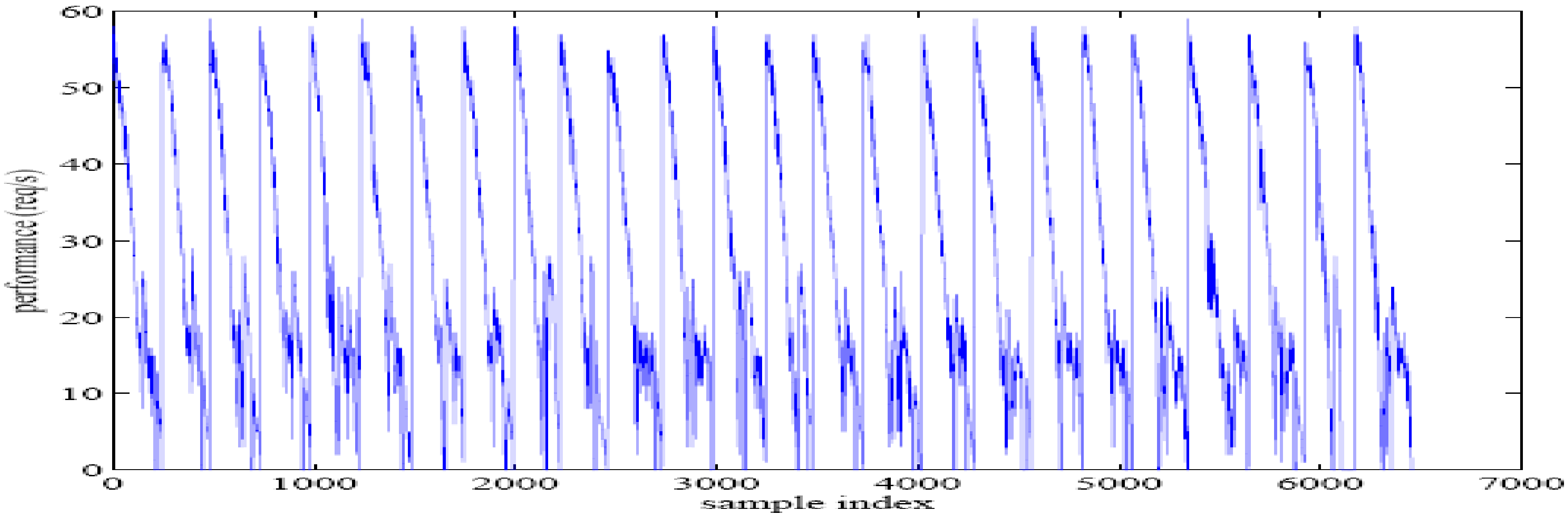
Inputs and Preprocessing

- We have two raw inputs
  - last observed performance
  - number of served requests since last rejuvenation
- From these, we compute (hundreds) of attributes
  - moving averages, their differences, filters, ..
- Finally, the most significant attributes are selected from this pool
- Those are used as final classifier inputs

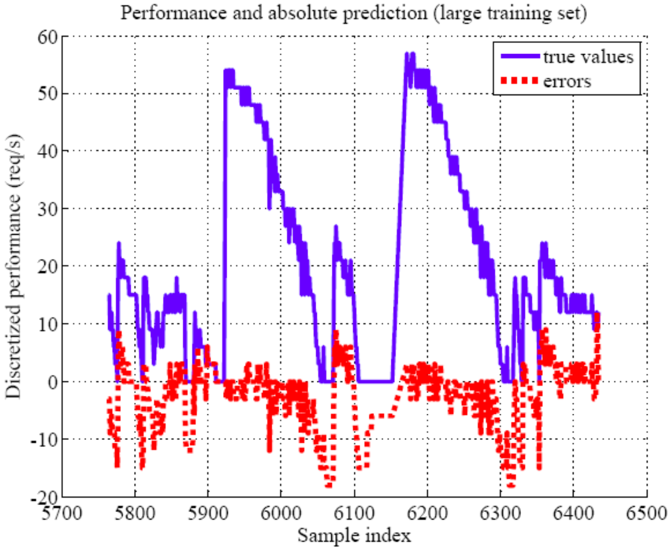


Prediction Experiments

- We predict the performance of a TPC-W benchmark (Java)
- Artificially inserted memory leaks per request
  - random leak size: 1-100 kBytes
- Data collected over 25 rejuvenation cycles
- Lead time: ι = 0, 2.5, 5, 12.5, 25, 50 minutes into the future
- Prediction model is rebuilt every 1000 samples (= 500mins)



Prediction Results

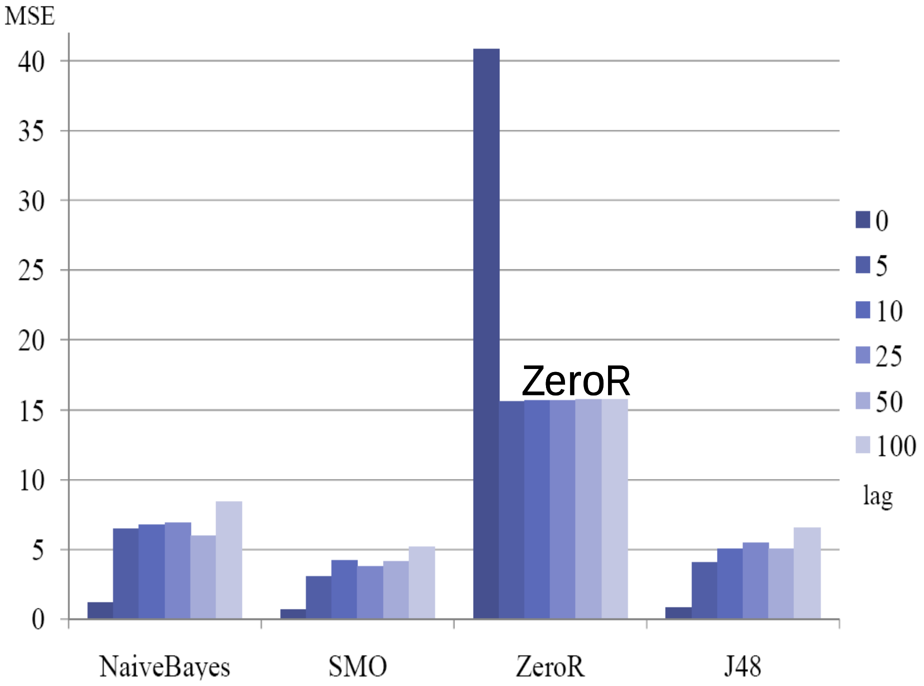


Algorithms

- No big differences among sophisticated algorithms and all lags > 0
- Again: more important are inputs, input preprocessing & selection
- ZeroR: "no brain" algorithm is significantly worse

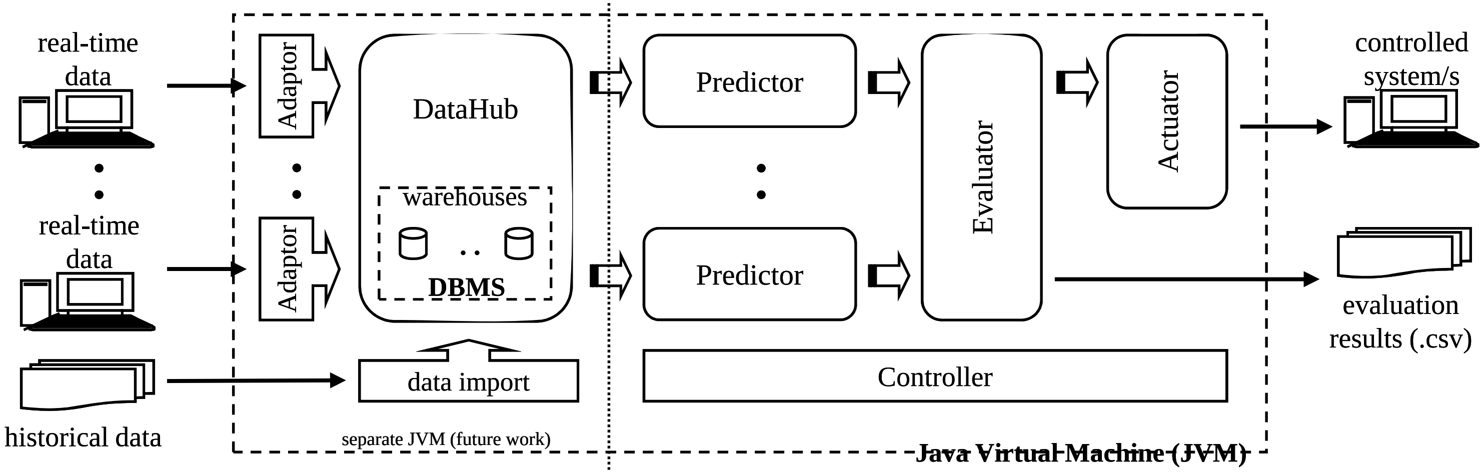
Accuracy

- Quite good: see figure
- blue: original (discretized) performance
- red: prediction error (5 min)



Mining Data Streams On-Line

- In real life, you want to have on-line prediction
  - Problem: most libs/frameworks allow only off-line studies
- We have developed StreamMiner, a Java framework
  - Allows for analysis/prediction of data streams in off-line and on-line modes (switching without code changes)
  - Built-in process of attribute generation, selection, signal prediction, model updates, and evaluation
- If you have a use case or just want to try it, contact me



Future Work

1. Add further inputs (e.g. OS metrics) and preprocess differently
   - measure more inputs & attributes
   - make a comparative study
2. Can we fix the applications instead of curing symptoms?
   - E.g. in Java LeakBot (IBM technology) can automatically identify leaking objects
   - best: automatically, no source code needed
3. Can we rejuvenate stateful applications as well?
   - transparent checkpointing needed
   - what if a part of the "deteriorated" state is checkpointed and restored?

Thank you.