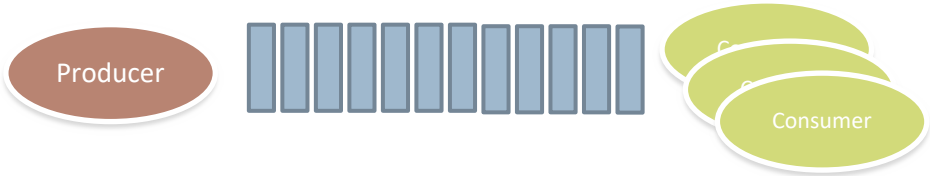
Leibniz Universität Hannover

Why Multi-Threading Should No Longer Be a DIY Job

Jannes Timm, **Jan S. Rellermeyer**

Concurrency is Hard…



?

Hardware

etc.

Software

To Thread or not to Thread?

(Single-Threaded) Event Driven

[*V.S. Pai, P. Druschel, W. Zwaenepoel: Flash: An efficient and portable Web server. In USENIX ATC 1999*]

[F. *Dabek, N. Zeldovich, N., F. Kaashoek, D. Mazieres, & R. Morris: Event-driven programming for robust software. In EuroSys 2002*]

Multithreaded

[*R. Von Behren, J. Condit & E. Brewer: Why Events Are a Bad Idea (for High-Concurrency Servers). In HotOS 2003*]

Thread Pools (reusing threads, sometimes adaptive using a Watermark model)

JVM (Executors), MariaDB, .NET CLR, etc.

[*F.W. Burton & M.R. Sleep: Executing functional programs on a virtual tree of processors. In FPCA 1981*]

Resource-Aware Threads

[*R. Von Behren, J. Condit, F. Zhou, G.C. Necula & E. Brewer Capriccio: scalable threads for internet services. ACM SIGOPS OSR 2003*]

State of the Art

Typical setup:

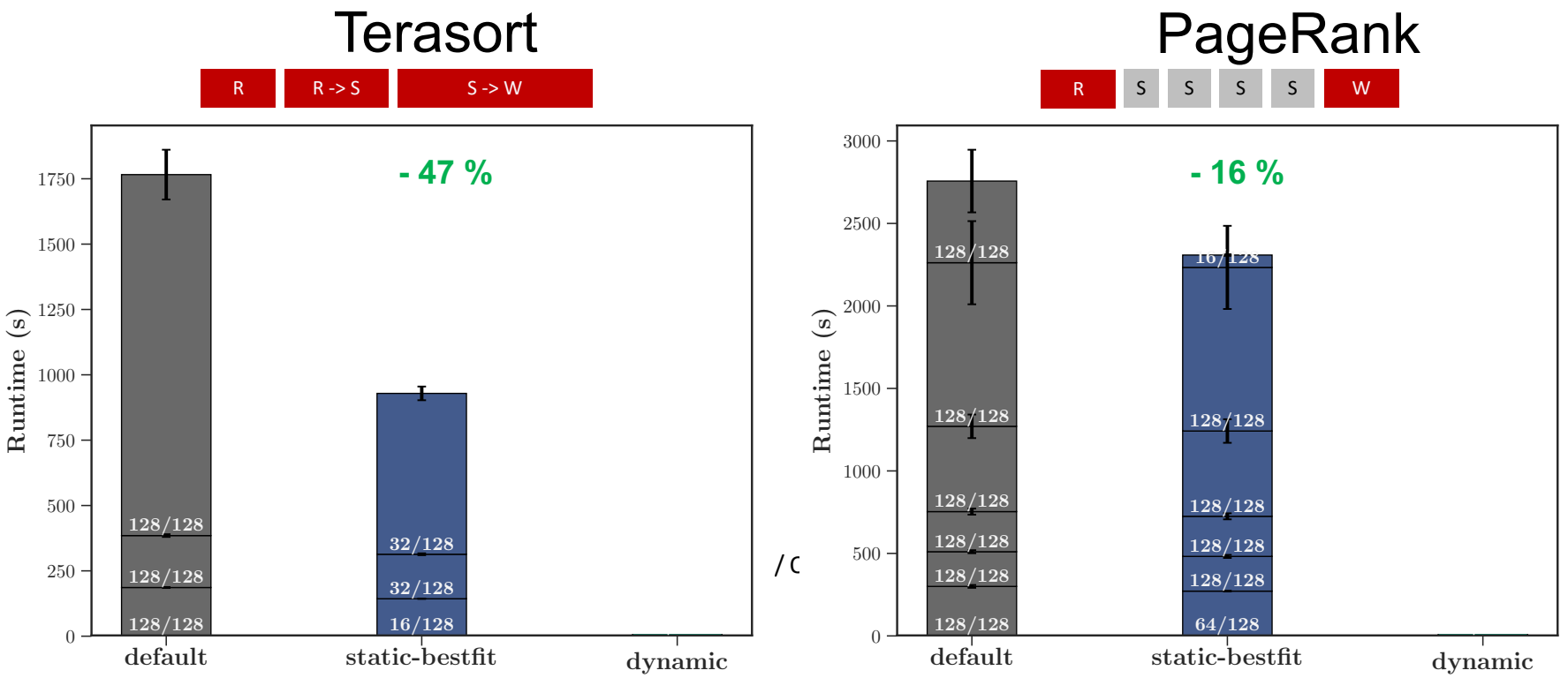
Hardware

#threads = #cores

flawed!

\+ manual tuning knob

When Less is More

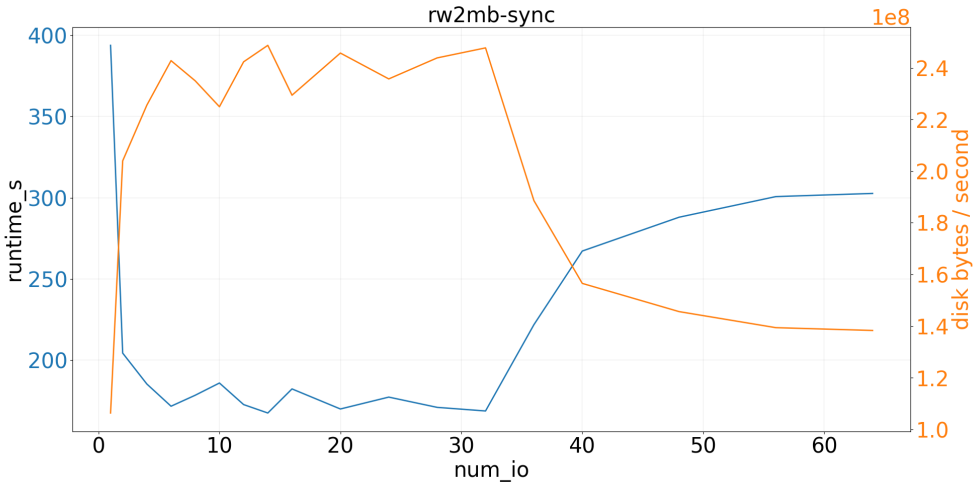


[*S. Omranian Khorasani, J.S. Rellermeyer, D. Epema: Self-Adaptive Executors for Big Data Processing. In: Middleware 2019*]

Towards Self-Adaptive Thread Pools

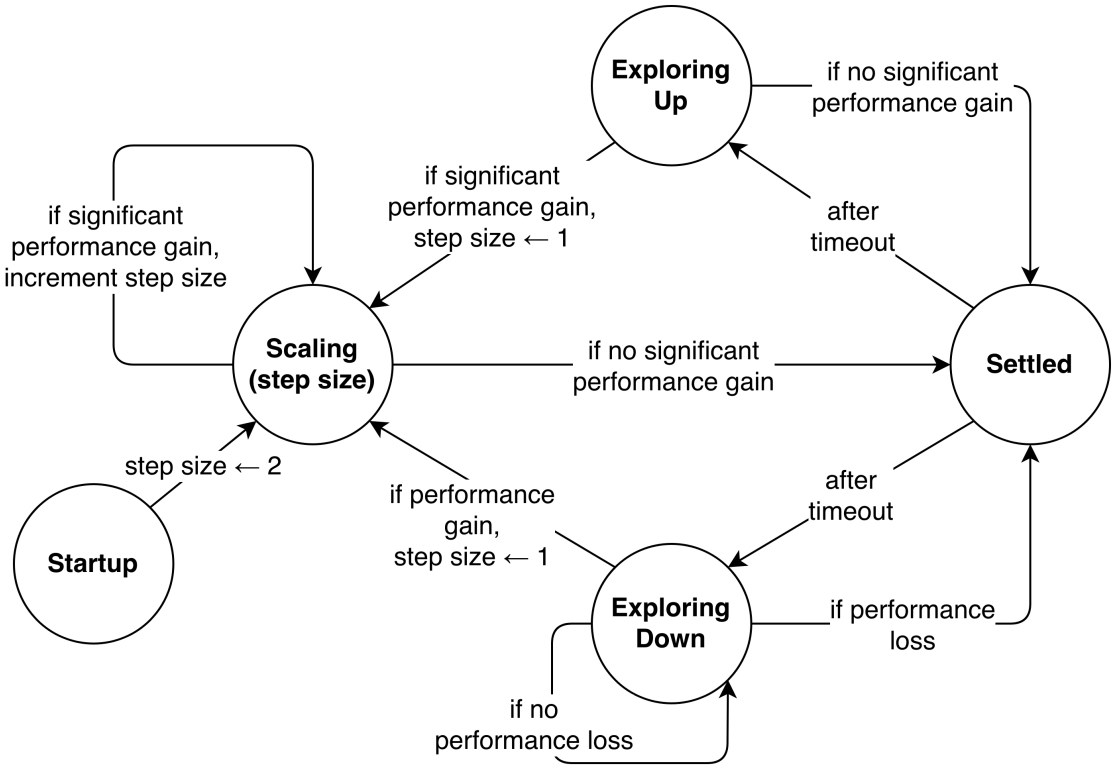
- Target Metric

rwchar-rate $\mathrm{R}[t_1, t_2] = \sum_{w \in W} \frac{(rchar_{w,t_2} + wchar_{w,t_2}) - (rchar_{w,t_1} + wchar_{w,t_1})}{t_2 - t_1}$



Controller Logic

- Hill-climbing approach



For most experiments: 1500ms interval length and 10% adjustment

Experimental Results: Synthetic Workload

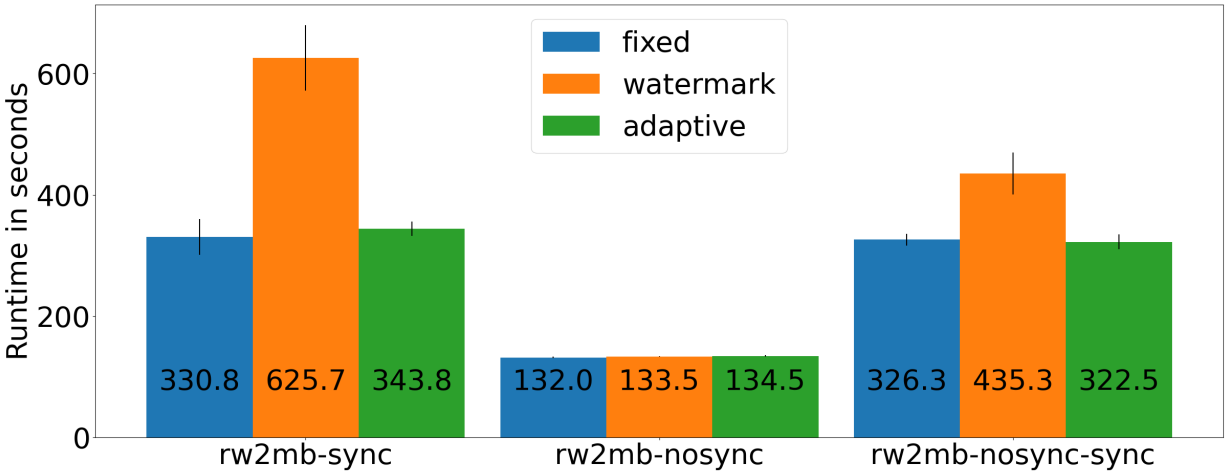
- Workers read a file of 2MiB into memory and write it back to disk ("*nosync*")
  - Variant "*sync*": writeback, call fsync
  - Variant "*nosync-sync*" first batch *nosync*, second batch *sync*

- Compare three setups:
  - "*fixed*": experimental optimum, determined through parameter sweep*
  - "*watermark*": classic high-low watermark model
  - "*adaptive*": our solution using hill-climbing with rwchar-rate

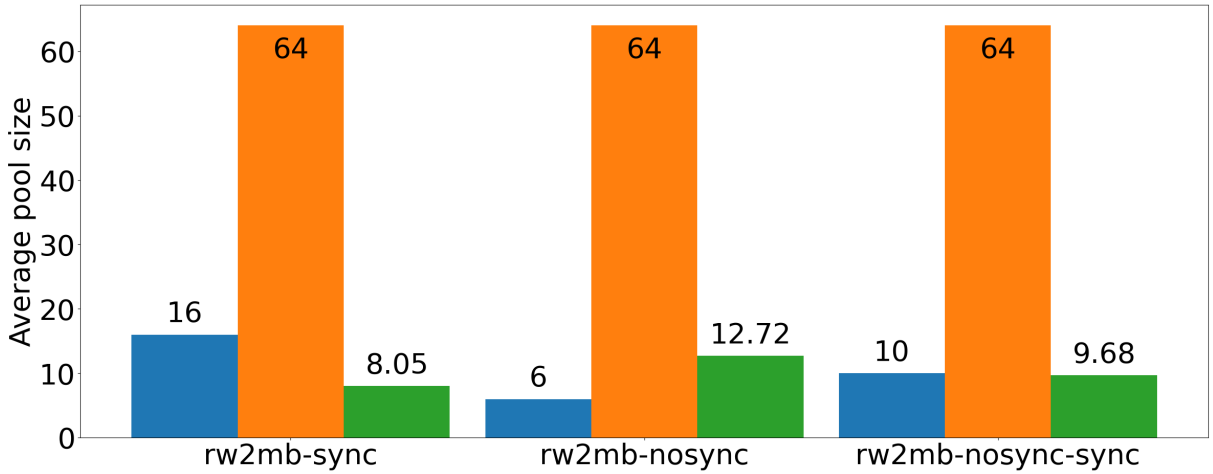
*3% tolerance and favor lower thread pool size

Experimental Results: Synthetic Workload

- Runtime:

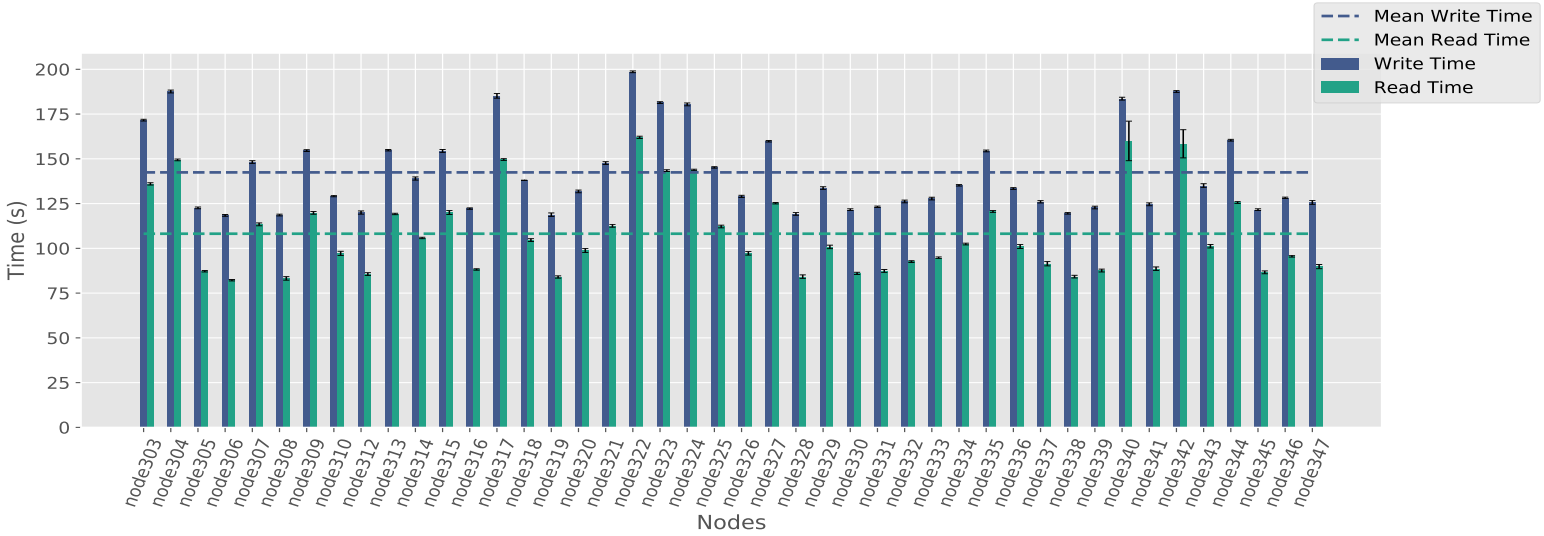


- Pool size:



Performance Variability Happens…

- In the small:

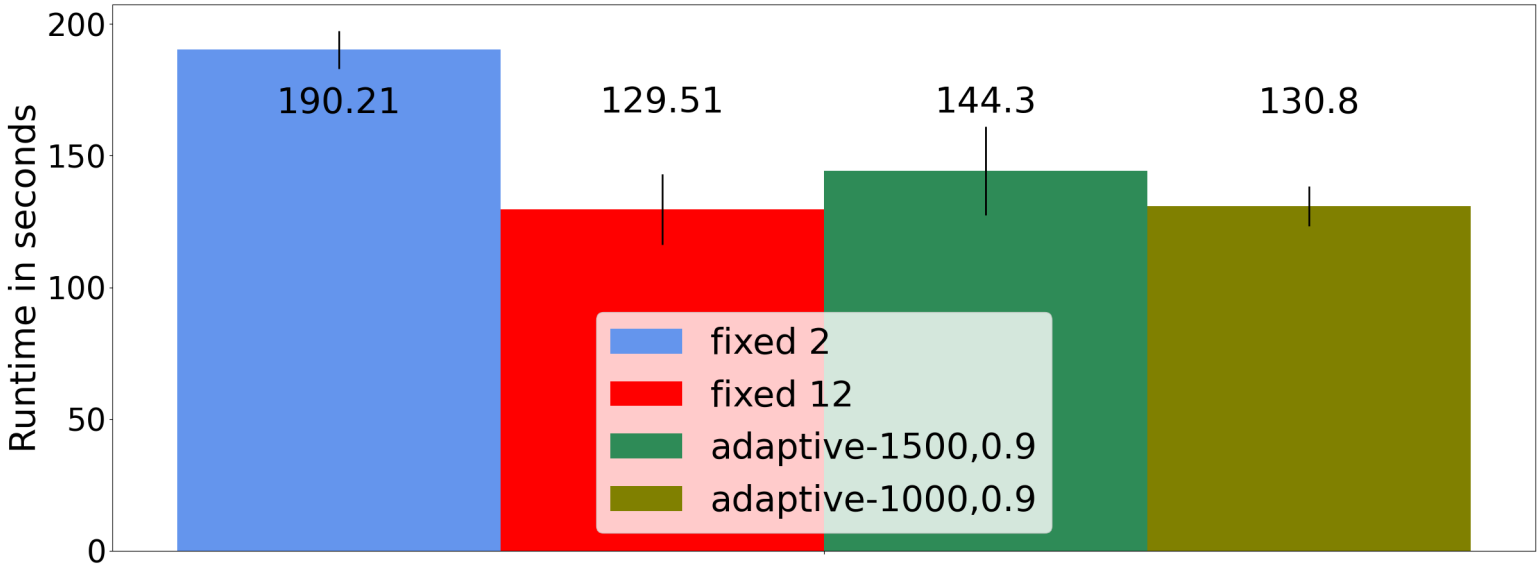


- In the large:

[*A. Uta, A. Custura, D. Duplyakin, I. Jimenez, J.S. Rellermeyer, C. Maltzahn, R. Ricci & A. Iosup: Is Big Data Performance Reproducible in Modern Cloud Networks?* In NSDI 2020]

Experimental Results: RocksDB

- Write-heavy workload: sequential key insertion (fillseq)
- Flush pool size has significant influence on the performance
  - Blue: default setting, Red: hand-optimized
- Interval length has an impact on the adaptive solution.
  - Green: 1500 ms, Olive: 1000 ms



Why Thread Pool Management Should be an OS Service

- We can use the machine resources more efficiently when tuning thread pools for I/O-intensive applications

- Manual thread pool tuning is tedious…
- … but also futile if the environment changes
  - Multi-tenant environments like containers, etc.
  - Statically tuned applications are selfish

- The OS already collects performance metrics and is supposed to be a mediator

- Providing thread pools as an OS service would allow us to implement better QOS for multi-tenant environments