The Case for Always-On Self-Learning Monitoring/Profiling of High Performance Transaction Processing Apps in Production

Pranta Das

Architect, AppDynamics, Inc.

pdas@appdynamics.com

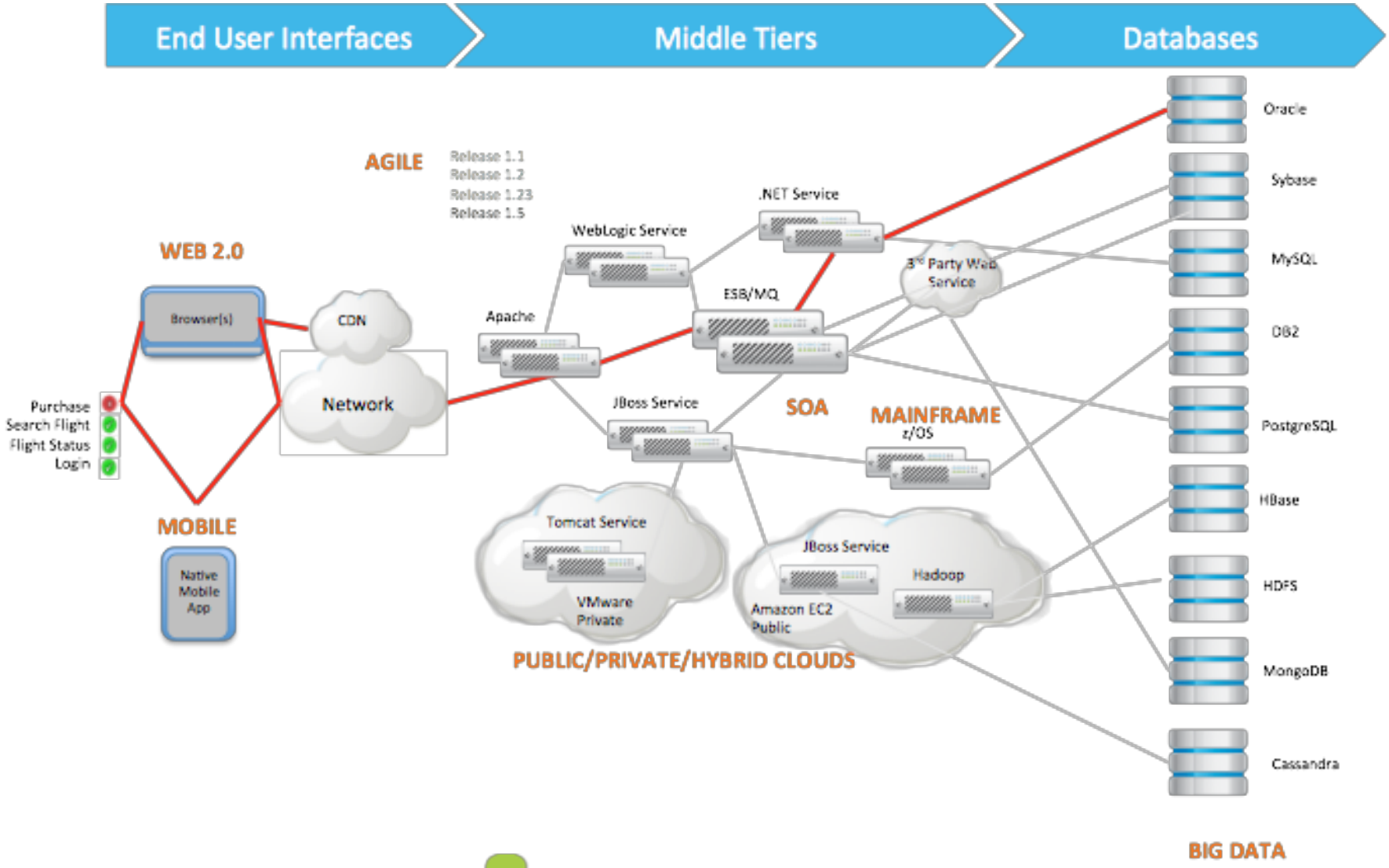
AppDynamics

15th International Workshop on High Performance Transaction Systems (HPTS)

September 22nd – 25th, 2013

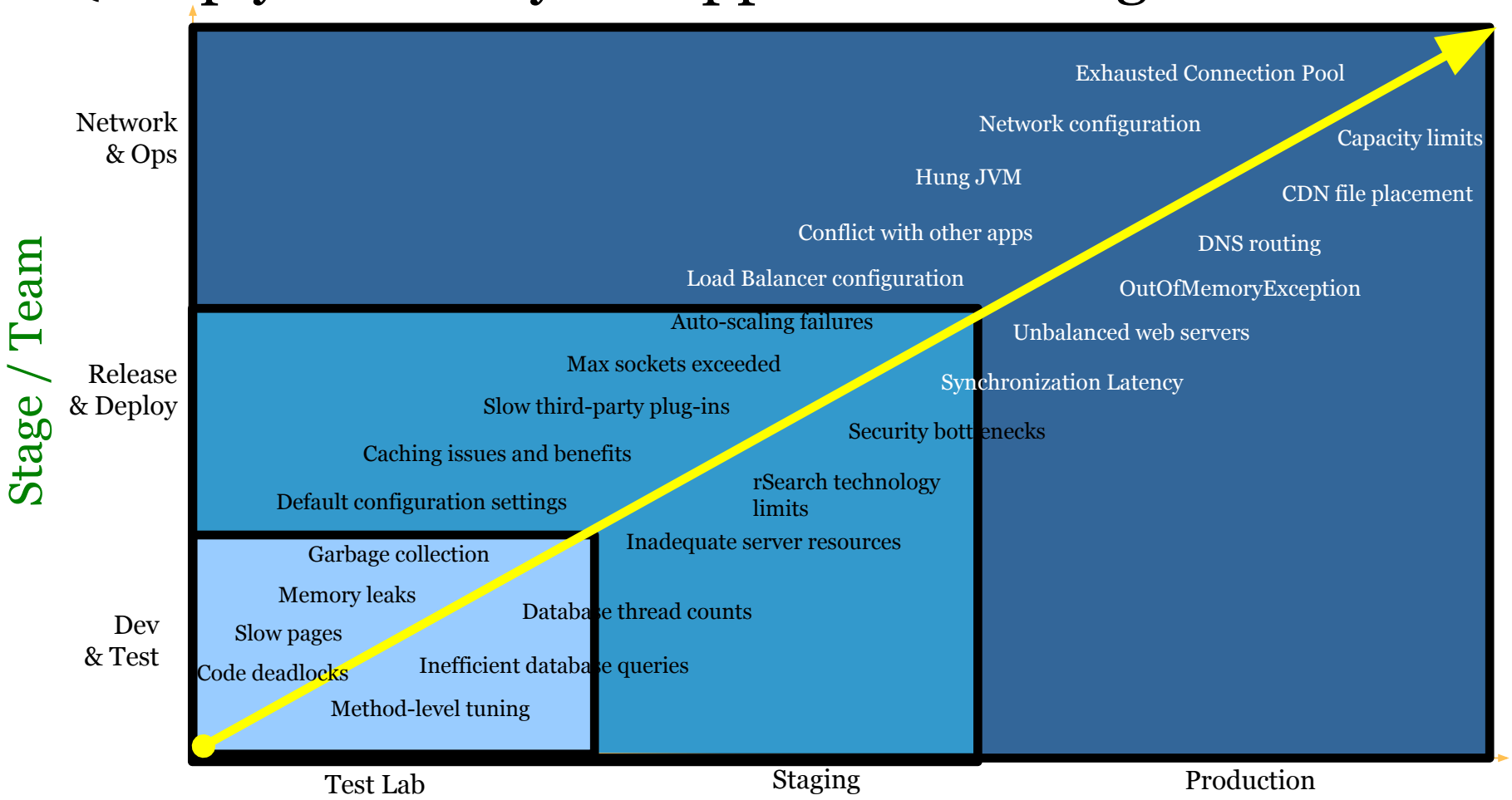
http://www.hpts.ws

Distributed Apps are getting more and more complex
And there's an App for everything!



AppDynamics

Most Performance problems manifest in Production (simply because your app lives the longest time there).



source: AppDynamics/SOASTA joint study, Nov. 2011

AppDynamics

Impediments to enabling monitoring of applications in production?

- Fear of performance overhead
- Most application development shops lack the discipline or the expertise to manually instrument their code, in an efficient and minimally intrusive manner
- Most monitoring systems require extensive configuration and do not adapt dynamically to production environments especially in the cloud
- Monitoring is quite often an afterthought when the proverbial sh*t hits the fan

So, what's changed?

- Advent of new technologies such as Byte-Code Instrumentation (BCI), especially in managed runtimes, such as:
  - Java Virtual Machine (JVM),
  - Microsoft .NET Common Language Runtime (CLR),
  - PHP and
  - Node.js
- Now, one can automate the instrumentation, and thus monitoring, of such applications without:
  - requiring any change from these applications,
  - incurring massive amounts of overhead

  and yet providing deep and wide visibility in production environments.

And what kind of scale are we talking about?

10,000 JVM
6.1B tx/day

5,000 CLR
500M tx/day

5,000 JVM
10M tx/day

3,500 JVM
7M tx/day

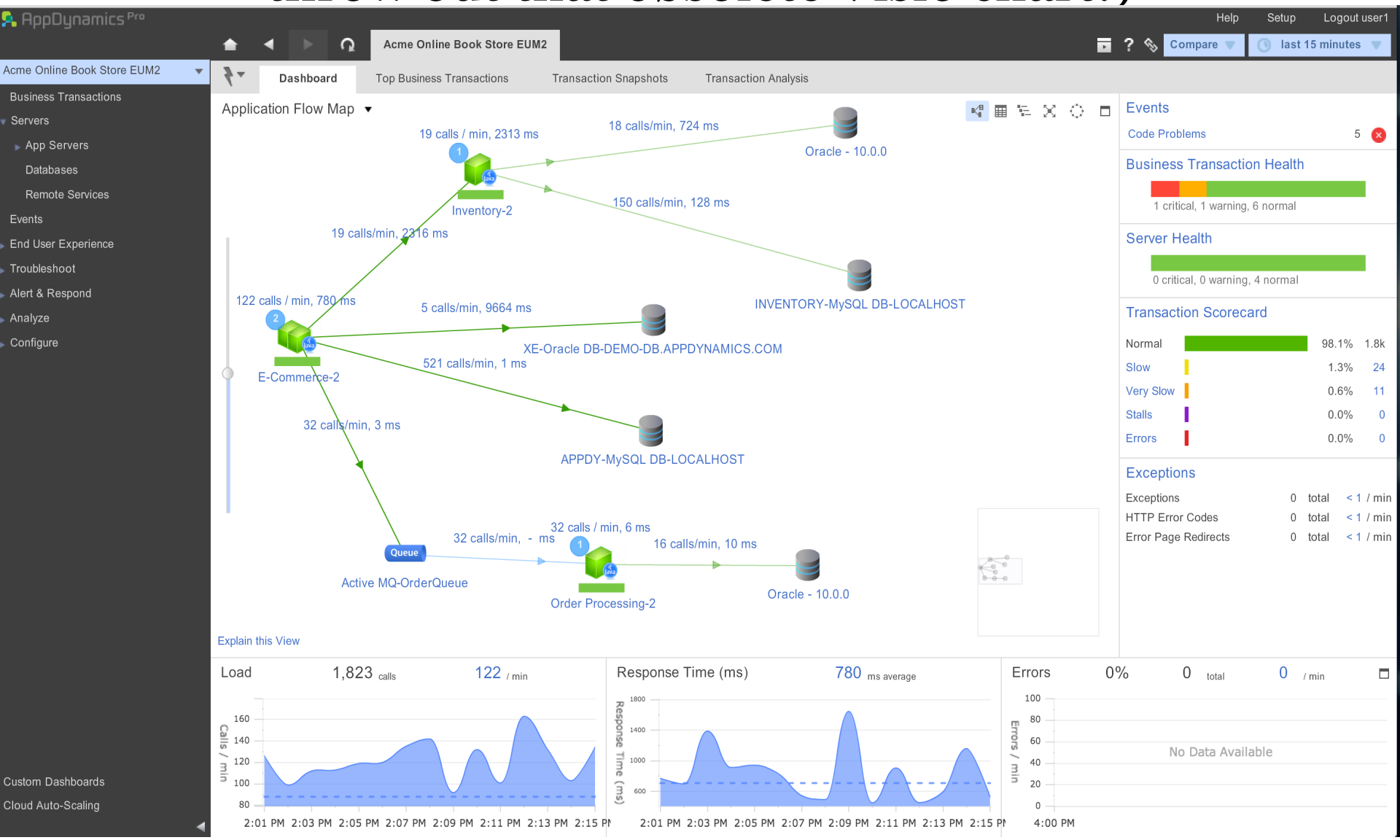
2,500 JVM
5M tx/day

2,500 JVM
10M tx/day

2,500 JVM
650k tx/day

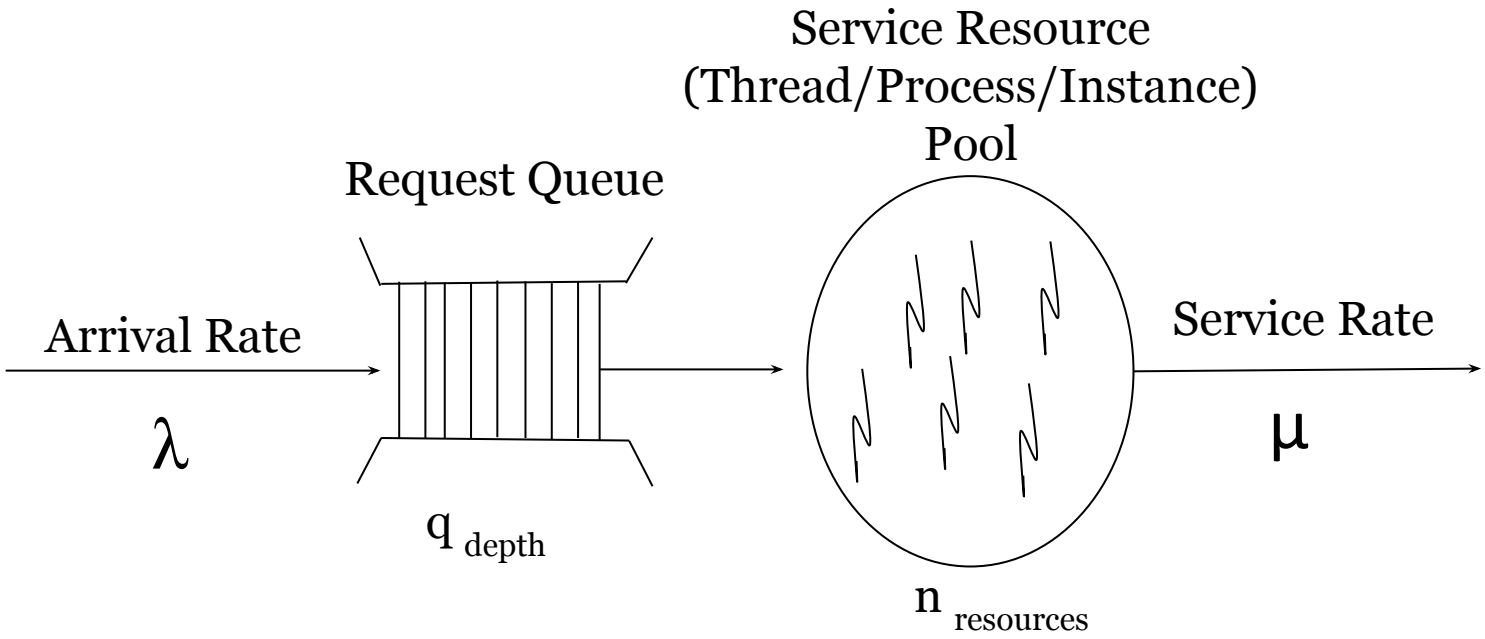
1,500 CLR
24M tx/day

Reverse Architecture – the Aha Moment! (Virginia, throw out that obsolete Visio chart!)



AppDynamics

Every Multi-tier Distributed App can be expressed as a Statistical Model of Traffic, Queues, Services and Resources



- For a given arrival rate λ, we need to maximize the service rate μ with an optimum value of $n_{resources}$. Monitoring q_{depth} will help us tune the application system to see if we need additional service compute resources to meet the current arrival rate.
- Having visibility into this data allows us not only to find bottlenecks in the code but also possibly flaws in design and architecture

Advantages of automated instrumentation of such applications over manual alternatives

- Capability of doing dynamic automated topology discovery (especially in elastic infrastructures such as public/private/hybrid clouds or virtualized environments)
- Automatic dynamic performance baseline establishment and automatic detection of outlier requests.
- Automatic cross-tier distributed correlation of requests, automated discovery of back-ends accessed by these applications such as SQL/NoSQL databases and Messaging middleware, release regression analysis (especially in Agile environments)
- Enabling policy-based automatic scaling (in cloud and virtualized environments) based on actual application metrics as opposed to just infrastructure metrics
- Promoting collaboration between Dev & Ops teams in troubleshooting performance problems in production systems.

Thanks!