Overcoming the Top Four Challenges to Real-Time Performance in Large-Scale, Data-Centric Applications

Tom Lubinski

Founder and CEO

SL Corporation

7 March 2012

- Disclaimers
- In 30 years, we’ve learned a lot
  (a grizzled veteran)
- But, we don’t know everything …
- … we could be wrong !
- My other computer is a Mac
- We have “shipped” …

Background

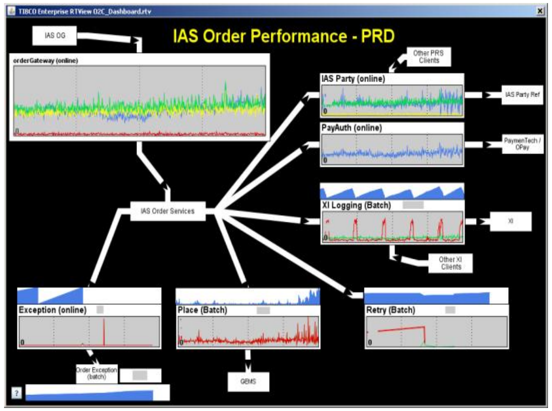
Extensive background in real-time process monitoring

Large volumes of dynamic data

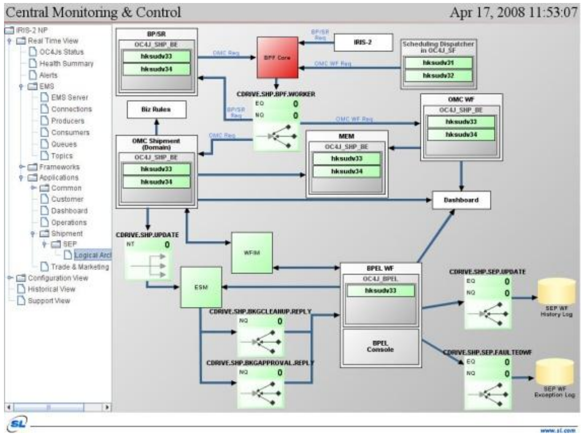
Visualization technologies

Mission-critical applications

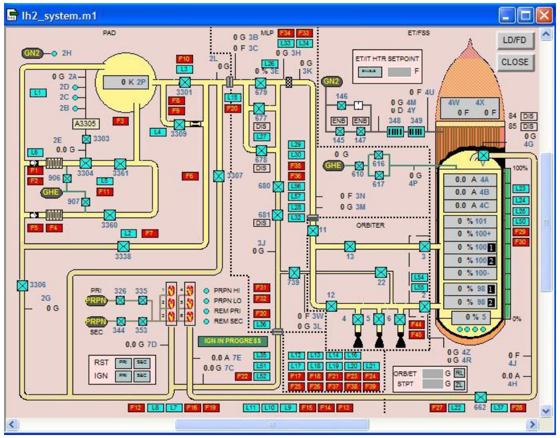
Connecticut Valley Power Grid Management System



Critical Tax Season Applications at Intuit



OOCL World Wide Shipment Tracking



NASA Space Shuttle Launch Control System

- Here to talk about Scalability and Performance

- Problem Space:

  Collection, Analysis, and Visualization in Real-Time of large volumes of monitoring data from large-scale, complex, distributed applications

  Emphasis: Real-Time, Large Volumes of Data

Challenges

- Challenge #1:

  Database Performance

  Common to see queries taking minutes

  How can you get real-time that way ?

Challenges

- Challenge #2:

  Network Data-Transfer Bandwidth

  Bigger pipes, but there’s more data to send

  How do you get the greatest throughput ?

Challenges

- Challenge #3:

  Processor Performance

  More cores just means more processes !

  How do you optimize your utilization ?

Challenges

- Challenge #4:

  Lack of Real-Time Predictability

  Virtualization is the new time-share !

  How can you trust your data ?

  "time-sharing", "network computer", "cloud", do things ever really change ?

- Solution – Clues ?

- Facts of Life:

Database – can’t live with it, can’t live without it

Network – it’s a funnel, no way around it

Processor – must limit what you ask it to do

Virtualization - it’s erratic, have to compensate

Solutions

- Solution #1:

  Proper Data Model

  - Data structures designed for real-time
  - In-memory structures to buffer database

Can your application be …

… like a high-performance racecar ?

What is most important part of racecar ?

(besides the engine)

… the Transmission …

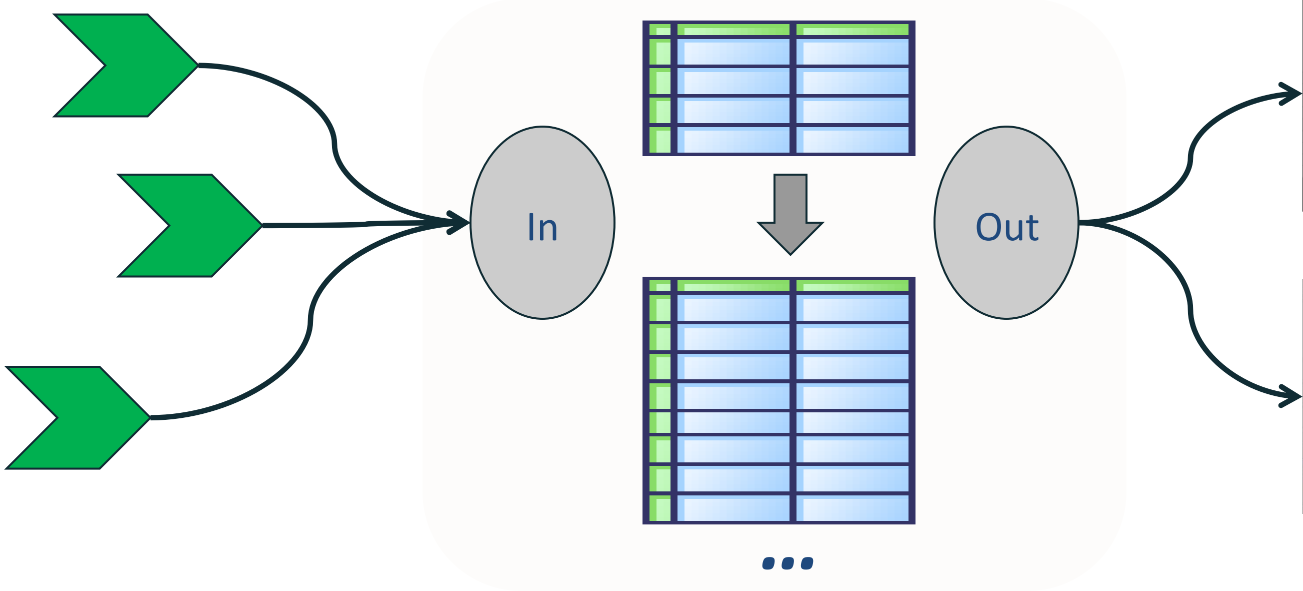
For Real-Time performance, it's the ***Cache*** *…*

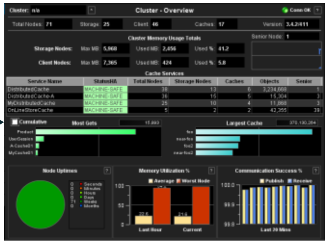
Not a simple "current value" cache

High-performance Real-time Multi-dimensional Data Cache

***Real-Time Cache –** optimized for performance* **!**

Current / History Tables:





Cluster: n/a　　**Cluster - Overview**　　Conn OK

Total Nodes: 71　Storage: 25　Client: 46　Caches: 17　Version: 3.4.2/411

Cluster Memory Usage Totals　　Senior Node: 1

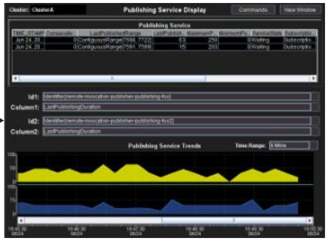
Storage Nodes: Max MB: 5,968　Used MB: 2,456　Used %: 41.2

Client Nodes: Max MB: 7,365　Used MB: 424　Used %: 5.8

Cache Services

Service Name	StatusHA	Total Nodes	Storage Nodes	Caches	Objects	Senior
DistributedCache	MACHINE-SAFE	38	13	6	3,234,668	1
DistributedCache-A	MACHINE-SAFE	36	15	5	15,304	3
MyDistributedCache	MACHINE-SAFE	25	10	4	11,868	3
OnLineStoreCache	MACHINE-SAFE	5	2	2	42,355	39

☐ Cumulative



Cluster: ClusterA　　**Publishing Service Display**　　Commands　New Window

Publishing Service

TIME_STAMP	Consecutiv...	LastPublishedRange	LastPublish...	MaximumP...	MinimumPu...	ServiceState	Subscriptio...
Jun 24, 20...	0	ContiguousRange[7698, 7722]	63	250	0	Waiting	Subscriptio...
Jun 24, 20...	0	ContiguousRange[7591, 7598]	15	203	0	Waiting	Subscriptio...

Id1: Identifier{remote-invocation-publisher-publishing-foo}

Column1: LastPublishingDuration

Id2: Identifier{remote-invocation-publisher-publishing-foo2}

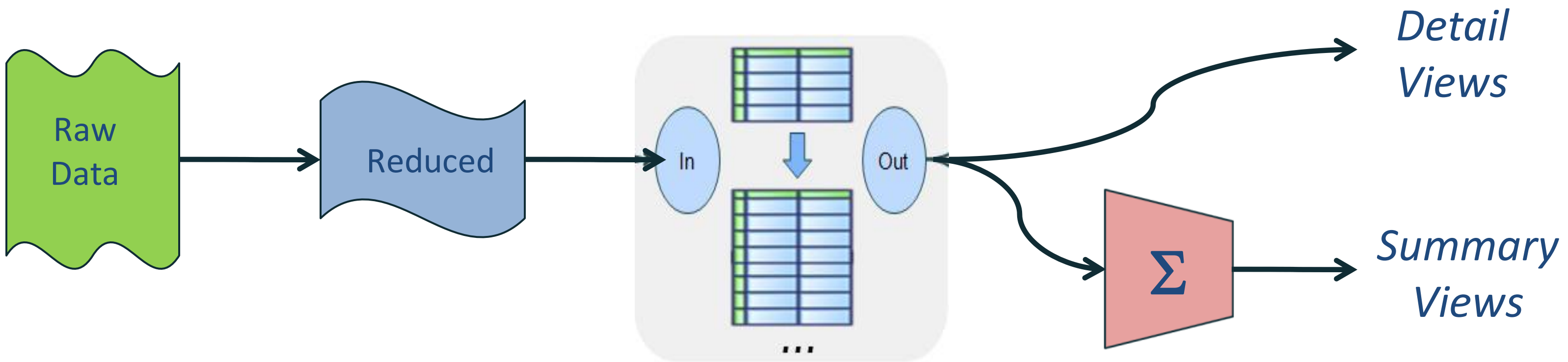
Column2: LastPublishingDuration

Publishing Service Trends　　Time Range: 5 Mins

Indexed Insertion -
asynchronous real-time data

Indexed extraction -
optimized transfer to clients

Real-Time Cache *– Data Processing / Aggregation*

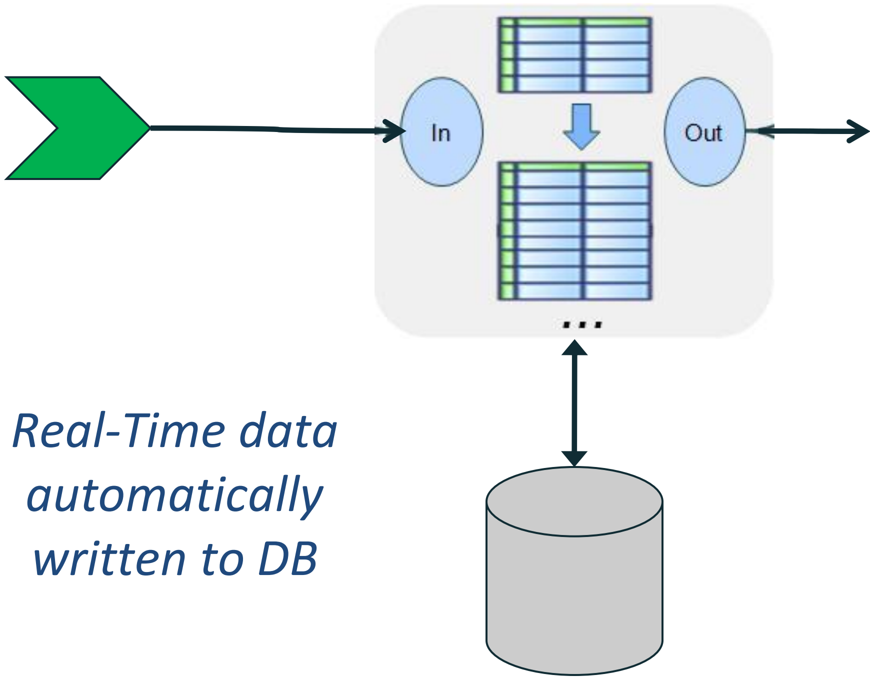


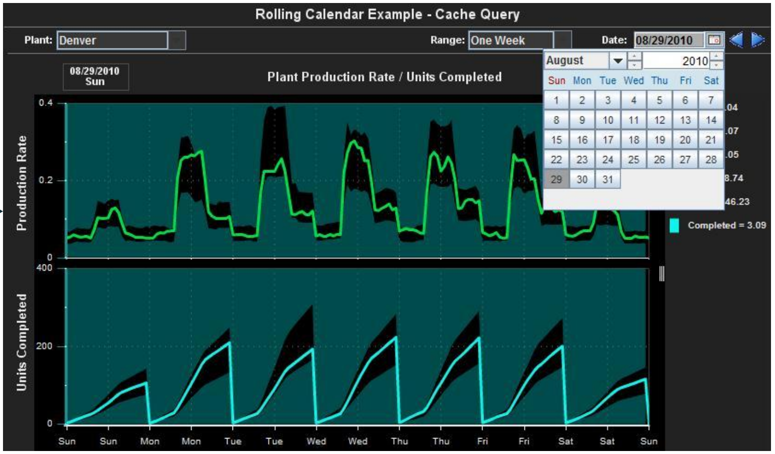
Reduction, Resolution, Aging

Aggregation

Real-Time Cache *– Database read/write through*

(optimized for timestamped multi-dimensional data)




Rolling Calendar Example - Cache Query

Plant: Denver

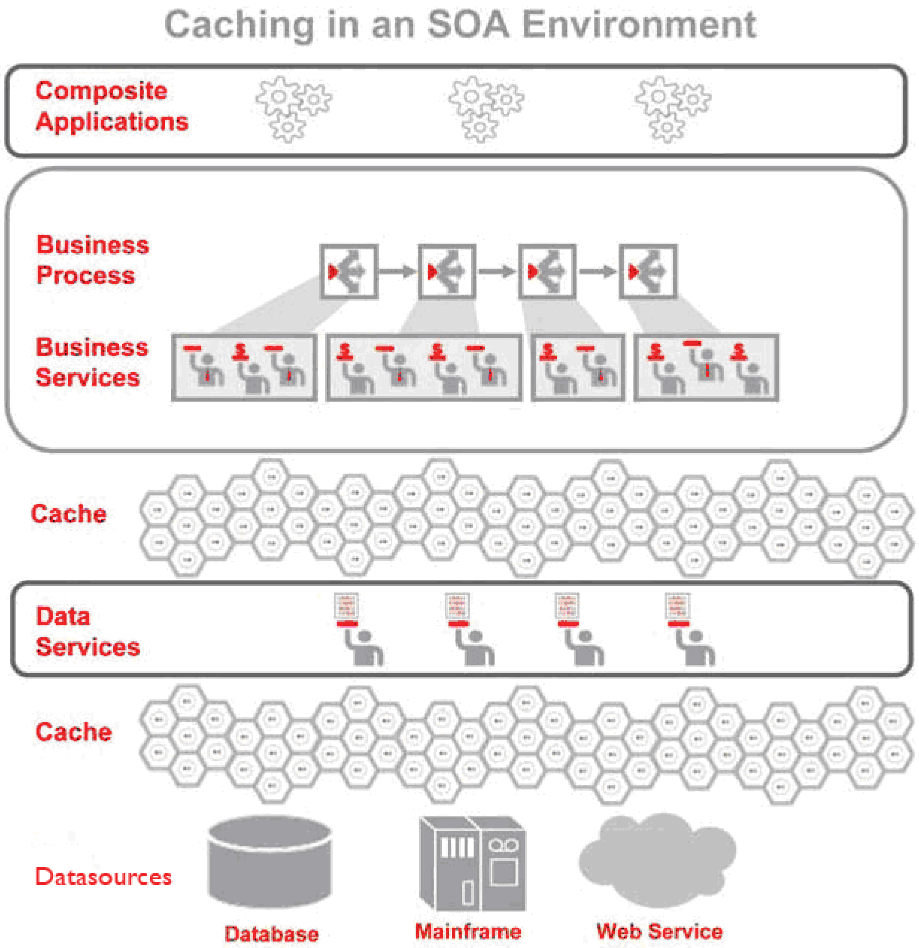
Range: One Week

Date: 08/29/2010

Real-Time data automatically written to DB

Seamless timeline navigation with automatic database query

This sounds a bit like Oracle Coherence …



Buffer database

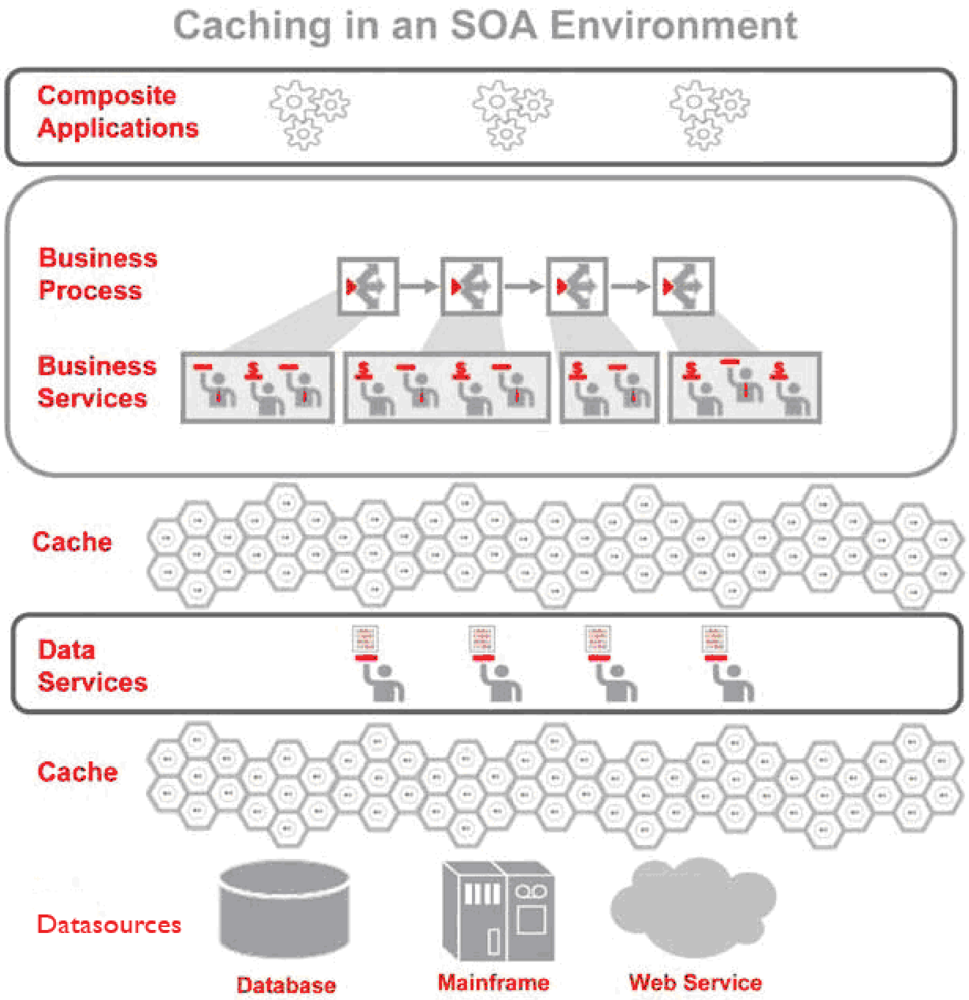
Read/write through

Listeners

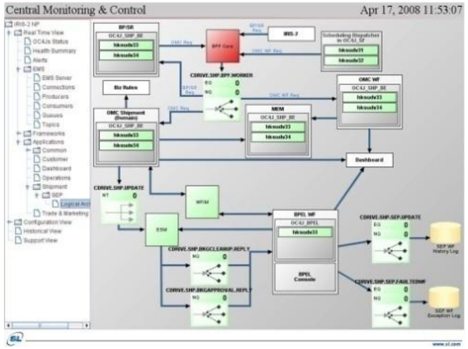
Indexed queries

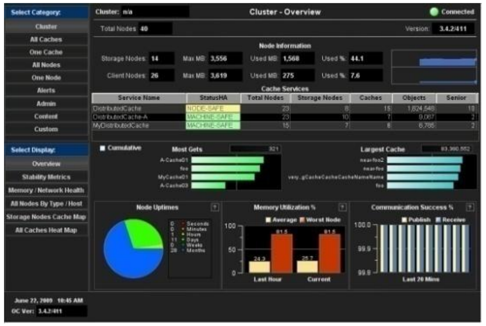
What's different ?

Multi-Tier Visibility into Monitoring Data



Unified Real-time display of data from all Application tiers





Cluster: n/a

Cluster - Overview

Total Nodes 40

Version: 3.4.2/411

Node Information

Storage Nodes: 14 Max MB: 3,556 Used MB: 1,568 Used %: 44.1

Client Nodes: 26 Max MB: 3,619 Used MB: 275 Used %: 7.6

Cache Services

Service Name	StatusHA	Total Nodes	Storage Nodes	Caches	Objects	Senior
DistributedCache	NODE-SAFE	23	8	15	1,824,548	10
DistributedCache-A	MACHINE-SAFE	23	10	7	9,067	2
MyDistributedCache	MACHINE-SAFE	15	7	8	6,785	2

☐ Cumulative

June 22, 2009 10:45 AM

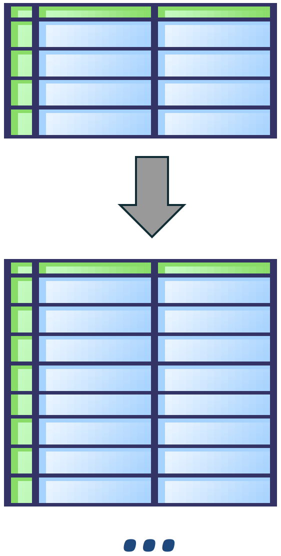
OC Ver: 3.4.2/411

In-depth Monitoring of Middleware Components

Different tools for different problems !

Real-Time Multi-dimensional data:

Current / History Tables:



Multiple rows (time range) of selected columns returned in one query

Coherence cache distributes objects (rows) = optimized horizontally

Real-Time multi-dimensional cache manages columns and optimizes vertically

Benefits: Indexed Real-Time Caching

Slow SQL queries minimized

Users shielded from database details

Minimize CPU load using effective indexing

Solutions

- Solution #2

  Server-Side Aggregation

  (am I being too obvious with this one ?)

  Know the use cases

  Joins and GroupBy done on server

  SQL does this, but do you need it ?

Problems with SQL Database Queries

Slow

Slowwer with concurrent queries

If you need it fast, it goes even slowwwwwwer !

SQL = Not portable

(Timestamps, especially)

Know your problem space !

Real-Time Monitoring:

Join and GroupBy heavily used

We wrote our own!

Performed in real-time on server-side data

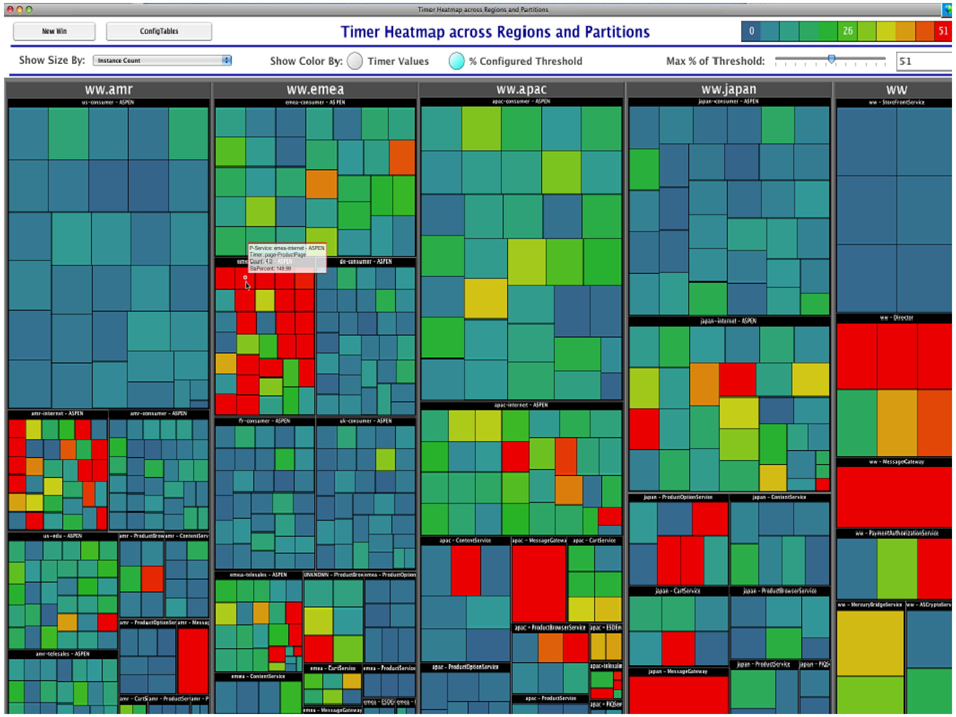
Optimized for real-time requirements

Display of Large Data Volumes

- Typical large implementation, distributed over several regions with many custom applications

- Heatmap View showing current state of entire system – size represents number of servers for application

- Color represents how close metric is to SLA – large red boxes are worst – drilldown to detail



Timer Heatmap across Regions and Partitions

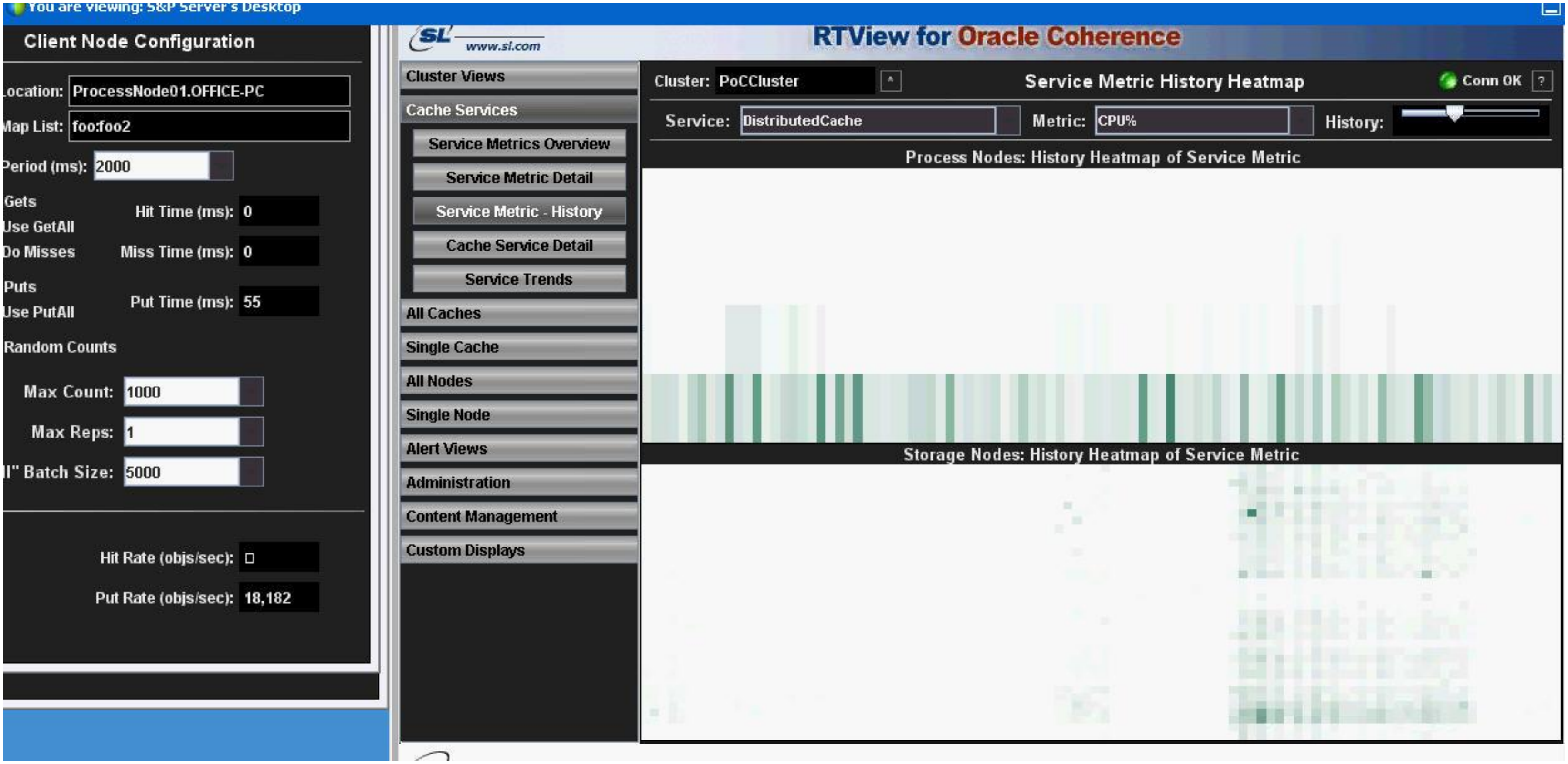
Show Size By: Instance Count

Show Color By: ○ Timer Values ● % Configured Threshold

Max % of Threshold: 51

Complex Visualizations of historical data

Observe “internal load balancing” of Data Grid



You are viewing: S&P Server's Desktop

Client Node Configuration

ocation: ProcessNode01.OFFICE-PC

Map List: foo:foo2

Period (ms): 2000

Gets

Use GetAll

Do Misses

Hit Time (ms): 0

Miss Time (ms): 0

Puts

Use PutAll

Put Time (ms): 55

Random Counts

Max Count: 1000

Max Reps: 1

I" Batch Size: 5000

Hit Rate (objs/sec): 0

Put Rate (objs/sec): 18,182

RTView for Oracle Coherence

Cluster: PoCCluster

Service Metric History Heatmap

Conn OK

Service: DistributedCache

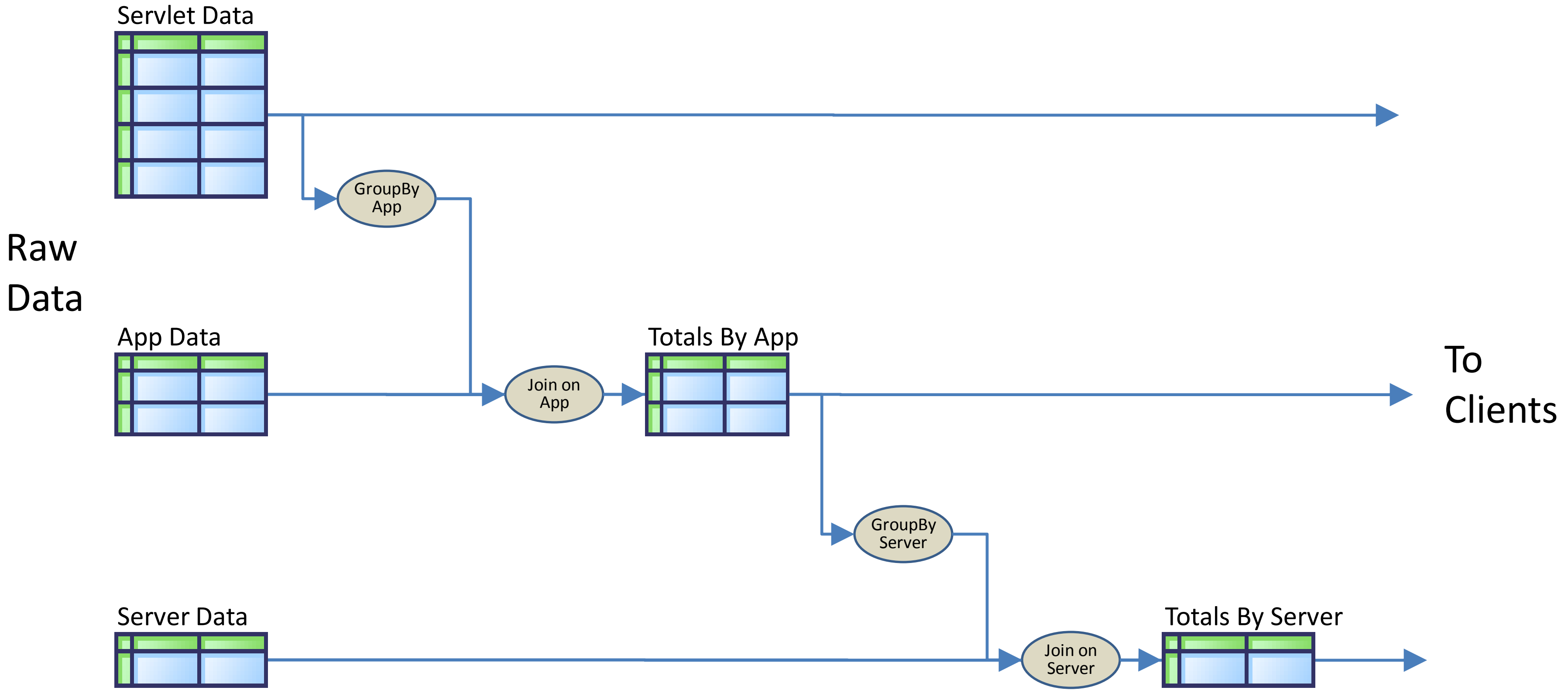
Metric: CPU%

History:

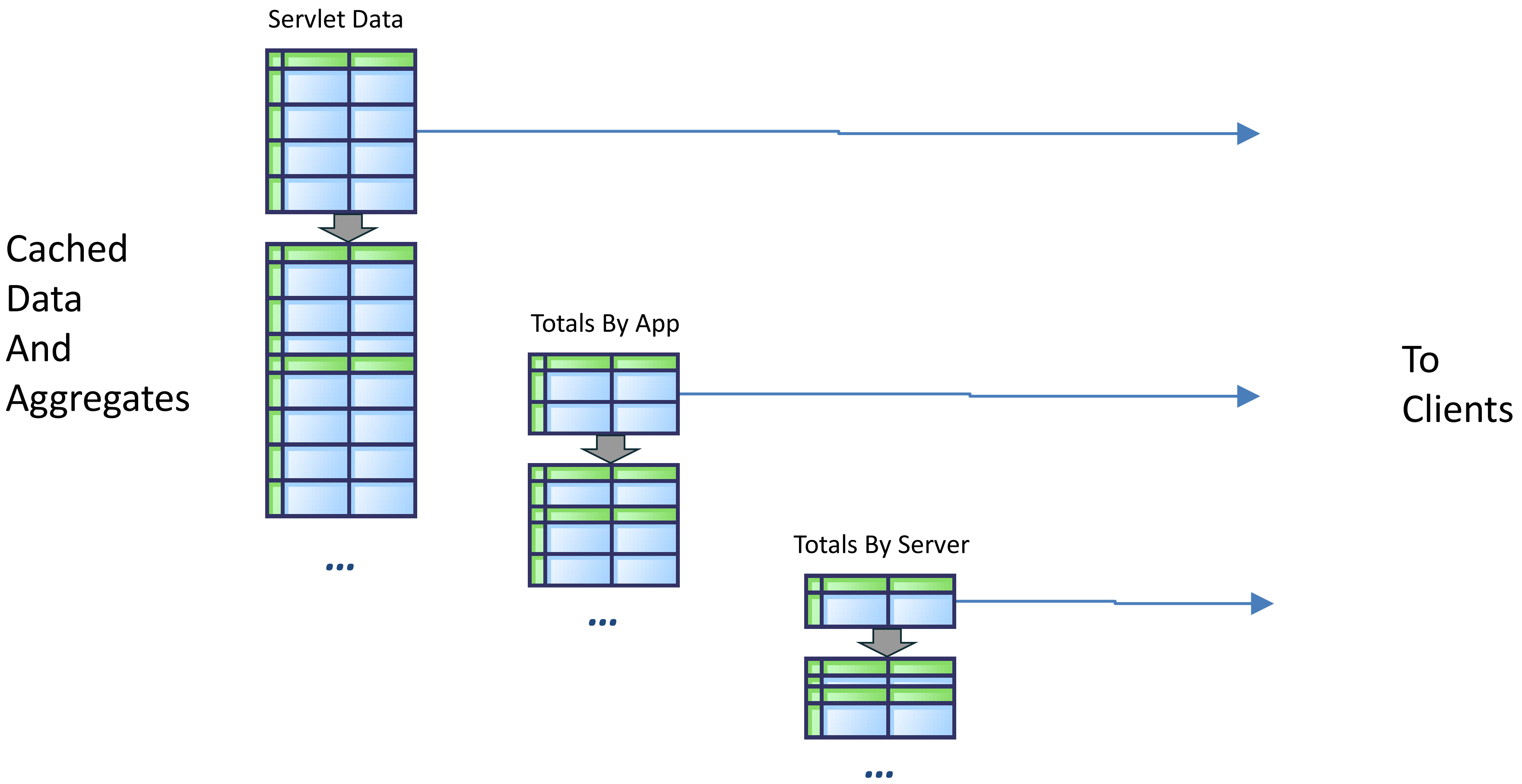
Process Nodes: History Heatmap of Service Metric

Storage Nodes: History Heatmap of Service Metric

- Example: Server-Side Aggregation/Caching



- Each cache can maintain its own history



- Result: trend chart of Totals by History has all data available immediately

- Using SQL would require:

  Query 3 tables

  2 GroupBys, 2 Joins, + Join on Timestamp (not portable)

Benefits: Server-Side Aggregation

Client requests and gets exactly what is needed

Client processing = zero
Server processing = done ahead of time
Current/History for aggregates readily available (No SQL)

Response time = fast

Solutions

- Solution #3

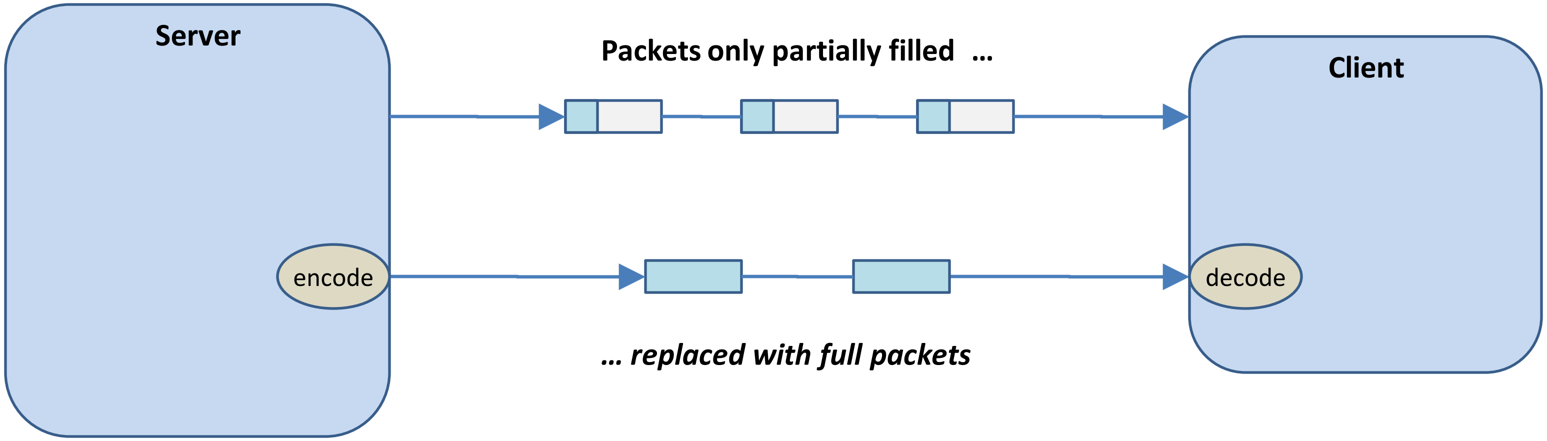
  Use Appropriate Design Patterns

  - Server-Side vs. Client-Side Processing
  - Efficient Data Transfer Patterns

- Pattern #1:

Data Compaction

(obvious, initial approach for any data transfers)

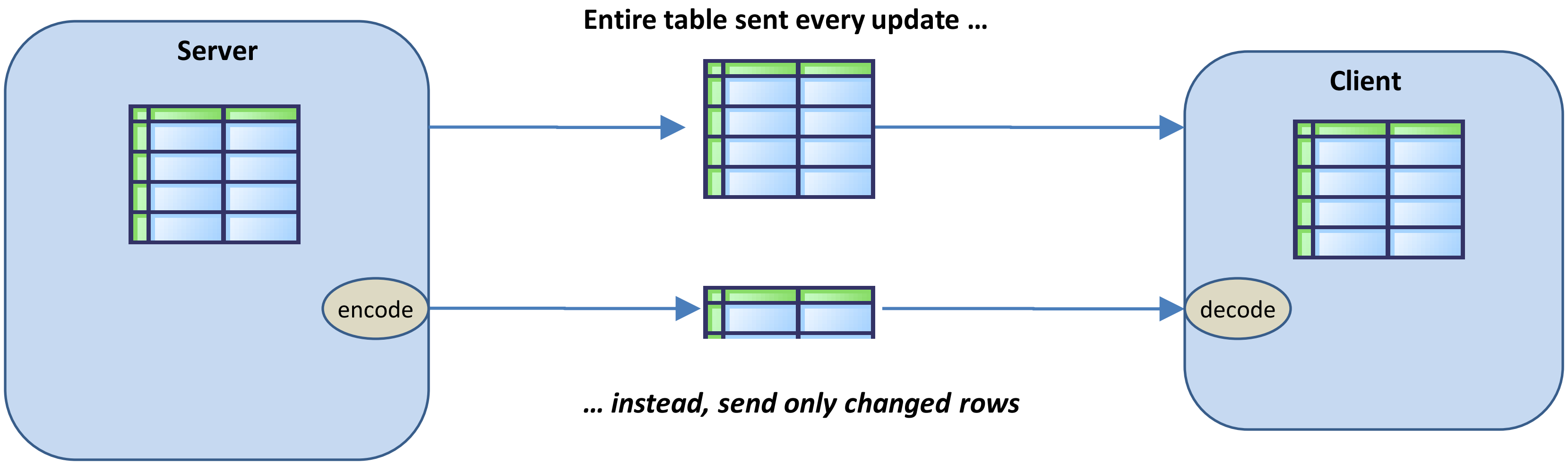


… even simple, non-proprietary algorithms can make big difference

- Pattern #2:

Data Current / Changed

(large data tables with sparse real-time updates)

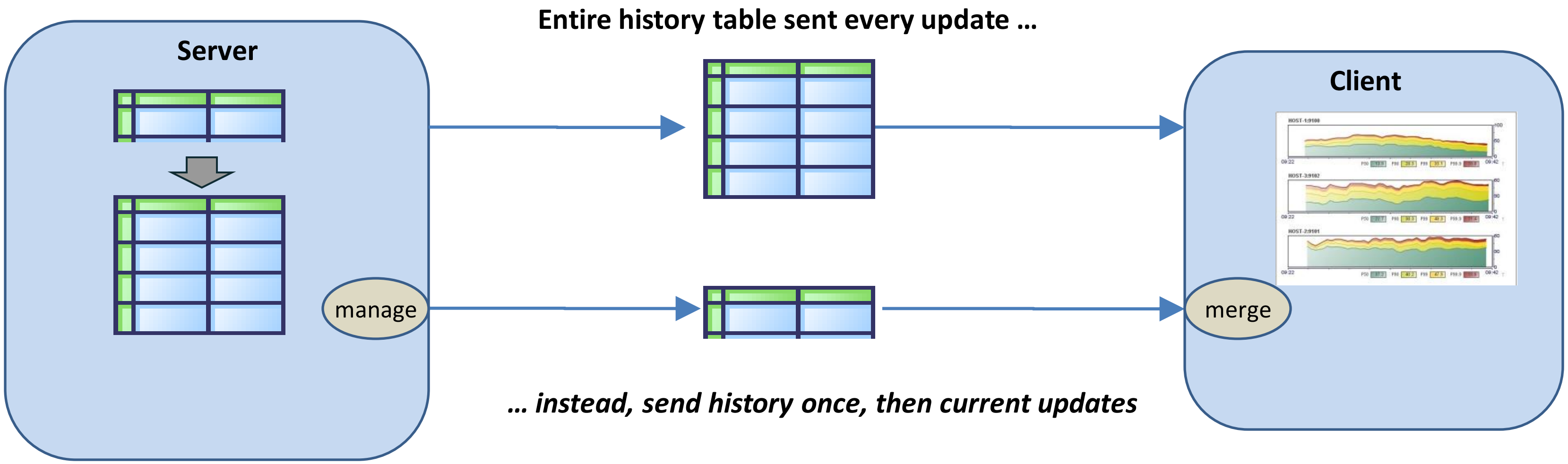

Entire table sent every update …
Server
Client
encode
decode
… instead, send only changed rows

… little more complex, requires indexing

- Pattern #3:

Data History / Current

(trend chart invoke with real-time updates)



Entire history table sent every update …

Server

manage

Client

merge

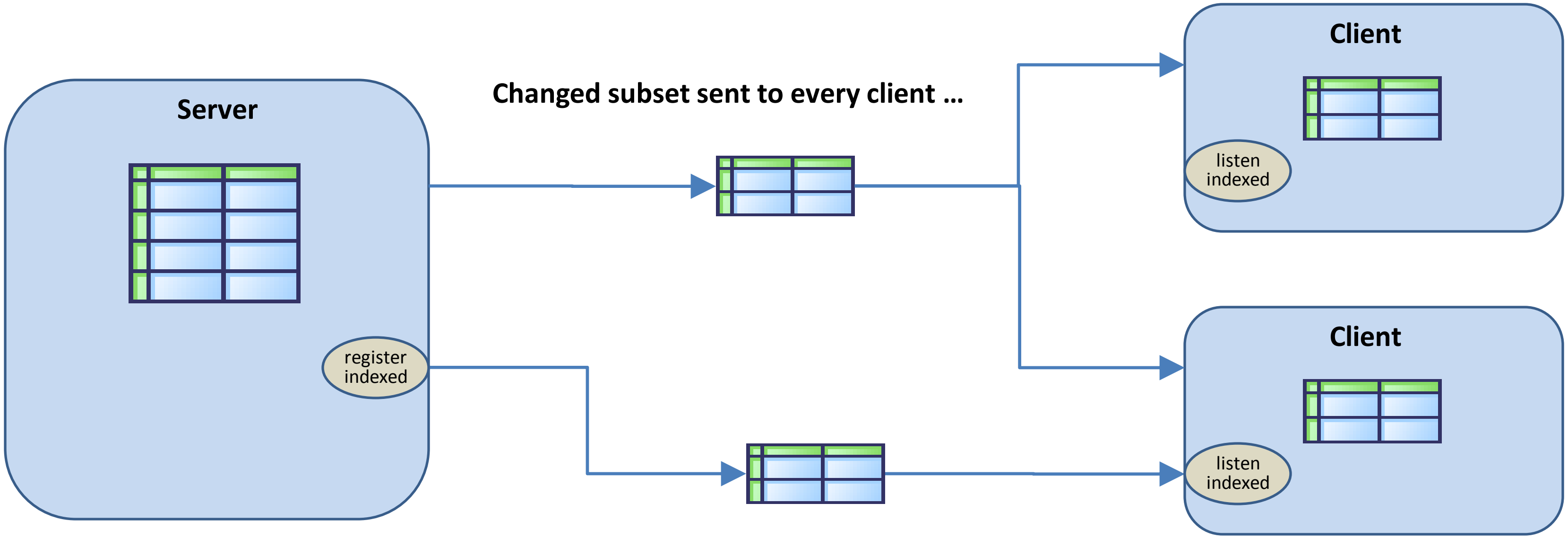
… instead, send history once, then current updates

… similar to current / changed pattern, but specific to history

- Pattern #4:

Data Current / Subset

(optimizing transfer of data subsets to multiple clients)



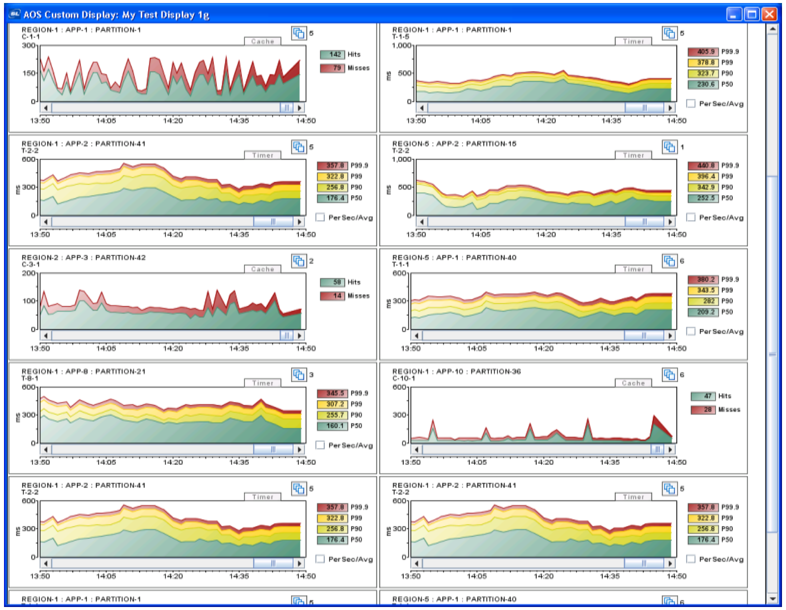
… instead, send subset only to registered client

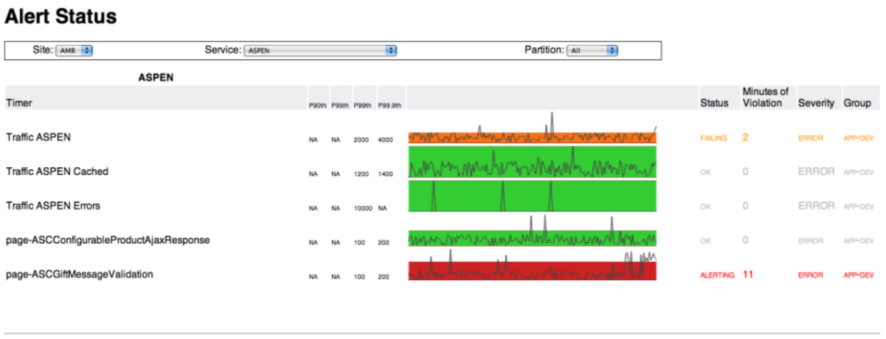
… requires registration logic coupled with cache

Drill-Down to Detail Metrics

- Drilldown to detail level metrics showing internal metrics from each application

- Sophisticated history and alert view with fine-tuning of thresholds for each metric


AOS Custom Display: My Test Display 1g
REGION-1 : APP-1 : PARTITION-1
REGION-1 : APP-2 : PARTITION-41
REGION-5 : APP-2 : PARTITION-15
REGION-2 : APP-3 : PARTITION-42
REGION-5 : APP-1 : PARTITION-40
REGION-1 : APP-8 : PARTITION-21
REGION-1 : APP-10 : PARTITION-36
Cache
Timer
Hits
Misses
P99.9
P99
P90
P50
Per Sec/Avg
ms
13:50
14:05
14:20
14:35
14:50



Alert Status

Site: AMR Service: ASPEN Partition: All

ASPEN

Timer	P90th	P95th	P99th	P99.9th		Status	Minutes of Violation	Severity	Group
Traffic ASPEN	NA	NA	2000	4000		FAILING	2	ERROR	APP-DEV
Traffic ASPEN Cached	NA	NA	1200	1400		OK	0	ERROR	APP-DEV
Traffic ASPEN Errors	NA	NA	10000	NA		OK	0	ERROR	APP-DEV
page-ASCConfigurableProductAjaxResponse	NA	NA	100	200		OK	0	ERROR	APP-DEV
page-ASCGiftMessageValidation	NA	NA	100	200		ALERTING	11	ERROR	APP-DEV

Benefits: Design Patterns for Data Transfer

Same problem over and over again solved similar way

Reduce load on network

Optimize response time – no unnecessary data

Conclusions

- Conclusion #1:

  Know your data !

  Data Model designed for real-time
  In-memory structures to buffer database
  Server-side aggregations

Conclusions

- Conclusion #2

  Respect Design Patterns !

  - Server-Side vs. Client-Side Processing
  - Efficient Data Transfer Patterns
  - Don’t over-generalize – solve the problem

Questions?

See www.sl.com
for more into about SL and RTView

Don’t miss SL Booth on Exhibit Floor !