Microsoft Financial Services Developer Conference

A Developer's Guide to Unleashing Your Data in High-Performance Applications

Distributed Caches

Marc Jacobs
Director
marcja@lab49.com

LAB 49 INNOVATION ON DEMAND

Synopsis

To make sure that you’re in the right room

LAB 49
INNOVATION ON DEMAND

Microsoft Financial Services
Developer Conference

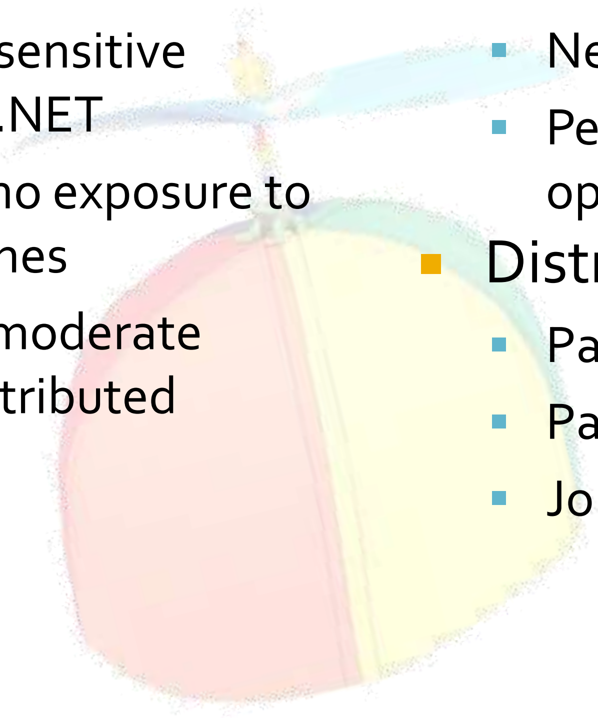
Agenda

- From the software developer’s view:
  - The basics of distributed caches
    - What are they? What services do they provide? What purpose do they serve? How do they function?
  - Use cases within financial services
    - What types of applications benefit from distributed caches? How can they be integrated in an architecture?
  - Performance evaluation and optimization
    - Methodology, performance variables, some results

Anti-Agenda

- Given there is more content than time, we will not cover:
  - Detailed vendor analysis and feature comparisons
  - Detailed inner architecture of distributed caches
  - Specifying, deploying, or managing caches
  - Reliability and fault tolerance features
  - Usage of advanced, product-specific developer features

Essential Demographics



YOU ARE:

- A software developer
  - …in financial services
  - …writing time-sensitive applications in .NET
  - …with little to no exposure to distributed caches
  - …with little to moderate exposure to distributed computing

YOU UNDERSTAND:

- Software Development
  - C# and .NET
  - Network services
  - Performance metrics and optimization
- Distributed computing
  - Parallelizing algorithms
  - Partitioning data
  - Jobs, tasks

Some Definitions

To ensure that we all are speaking the same language

Define:(High-Performance)

- Technique of computing that:
  - Unites many machines into a coordinated group
  - Uses the group to execute code
  - Balances workload across machines in the group
  - Manages the lifetime of tasks and jobs
  - Reduces bottlenecks on computation
- Loose synonyms:
  - Grid-, distributed-, cloud-computing, HPC

Define:(Distributed Cache)

- Server product that:
  - Unites many machines into a coordinated group
  - Uses the group to store granular data
  - Balances data across all machines in group
  - Diffuses I/O across all machines in group
  - Reduces bottlenecks on data movement
- Loose synonyms:
  - Data grid, data fabric

DCP ≈ Distributed Cache Product

- There isn’t a brief, widely used acronym for distributed cache products.
- So, here’s a makeshift one for this talk.

DCP

The Dual Purpose of DCP

SCALABILITY

- Ensure that an HPC system can tolerate:
  - Increases in client count
  - Increases in data sizes
  - Increases in data movement
  - Increases in concurrent apps
- Adds bandwidth and load-balancing to data sources

- To be discussed

RELIABILITY

- Ensure that an HPC system can tolerate:
  - Hardware failure
  - Network failure
  - Server software failure
  - Denial of service
- Adds self-healing and resilience to data sources

- To be omitted

HPC Drives DCP

- Though HPC has so far been adopted faster than DCP, they will necessarily rise together
  - Distinct best-of-breed products in DCP and HPC today are likely to coalesce in the coming years
- The reason :
  - Parallel clients deluge central data sources
  - Many hands make light work (but heavy load)
  - HPC creates a problem that DCP nicely solves

Analog Example

- An infinite line waits to get a pint of stout from a single tap
  - Process: get glass, pour body, let settle, pour head, serve glass, accept payment, make change, collect tip
- An extra bartender or two speeds service as they can serve multiple customers at once
  - But more creates contention at tap
- Adding more taps permit more bartenders with less contention and faster service
  - Parallelism only scales when both bartenders and taps scale

Some Use Cases

To present motivating examples within financial services

LAB 49
INNOVATION ON DEMAND

Microsoft Financial Services
Developer Conference

Cornucopia of Cases

- Financial services is replete with “delightfully parallel” apps, for example:
  - Portfolio Management
  - Fixed Income Pricing
  - Reporting
  - Simulation and Back-testing
- When run in parallel, these apps generate a lot of I/O, esp. from central data sources

Portfolio Management

A hedge fund manages a corpus of investment portfolios for its client accounts, each comprised of many asset types. Throughout each trading day, as markets open and close and as new market information is collected, portfolios are rebalanced, re-priced, and re-hedged.

```
foreach account:
  foreach market:
     calculate ideal exposure
     allocate ideal exposure
     calculate ideal trades
     adjust trades (cost/risk)
  adjust trades (cost/risk)
  hedge trades
```

Fixed Income Pricing

An investment bank furnishes pricing information to traders and counterparties for fixed income instruments. Throughout the trading day, market information is revealed that affects the pricing of fixed income instruments and structured products.

```
foreach bond:
  foreach path:
     foreach month:
       calculate dependencies
       calculate price
     aggregate price
  aggregate paths
```

Reporting

Financial services firms generate customized reports for both internal and external use. Whether daily, monthly, or quarterly, reports create significant demand for data.

```
foreach account:
  foreach report:
     foreach report element:
       calculate element
     calculate report
     format report
     export report
  assemble report package
```

Simulation and Back-Testing

Any financial services firm that does any level of quantitative analysis and modeling will require the ability to simulate models and back-test them against historical data. This often involves executing a model over many years' worth of data.

```
foreach configuration:
  foreach instrument:
     foreach period:
       calculate model value
       calculate model error
     aggregate model value
     aggregate model error
  aggregate model error
aggregate model error
```

Common Characteristics

Many Units of Parallelism

- Accounts
- Instruments
- Samples
- Paths
- Shocks
- Periods
- Parameters

Large Volume of Input Data

- Reference data
- Indicative data
- Parameters

Input Data Is Overlapping

- Reference data and indicative data, for example, required across many apps, not just one

Theme: Shared + Specific Data

SHARED DATA

- Examples:
  - Market quotes
  - Yield curves
  - Exchange rates
  - Reference data
  - Model parameters
- Characteristics:
  - Often bandwidth hot spots
  - Cache-friendly

SPECIFIC DATA

- Examples:
  - Account details
  - Portfolio allocations
  - Bond structure
  - Prepayment data
  - Intermediate state
- Characteristics:
  - Often server hot spot
  - Cache-insensitive

Theme: Fan Out → Fan In

- As we exploit parallelism to relieve workload bottlenecks on CPU, additional parallelism creates bottlenecks on central data sources
  - Non-replicated file systems, web servers, and databases get deluged in HPC scenarios
  - Clusters or load-balanced replicas of the above reach early scalability limits and create complexity in both IT management and software design
  - Read-Write patterns are particularly challenging

Scaling Topologies

Scaling Up

- Means getting bigger, faster hardware better able to tolerate the workload.
- Offers limited scalable processing power and memory, but no scalable bandwidth.

Scaling Away

- Means implementing opportunistic multilevel caching to protect critical resources from repeated reads.
- When clients repeatedly ask for the same pieces of data, information is returned from nearest cache instead of critical resources.

Scaling Out

- Means enlisting more machines working together to share in the workload.
- Offers unlimited scalable processing power and limited scalable memory and bandwidth

Enter the Distributed Cache

- Represents a combination of scaling up, away, and out techniques
  - Leverages the additive bandwidth, memory, and processing power of multiple machines
  - Offers a simplified programming model over replicated server or local copy models

Using Distributed Caches

To get up and running

Choices, Choices, Choices…

- A varied number of products at different price points and with different feature sets:

Product	Price	Size	Complex	Feature
Alachisoft NCache	$	&	!	*
GemStone GemFire	$$$	&&	!!	**
GigaSpaces XAP	$$$	&&&	!!	***
IBM ObjectGrid	$$$	&&&	!!!	**
Oracle Coherence	$$$	&&	!!	**
ScaleOut StateServer	$$	&	!	*

Our Pedagogic Example

- Introducing ScaleOut StateServer (SOSS)
  - Written in C/C++ for Windows, offers .NET API
  - Straightforward install (either server or client)
- Comparatively:
  - Offers similar interfaces and features as others
  - Offers more narrow focus on fast/simple DCP
  - Offers advanced features and fault tolerance support but that’s out of scope for this session

Essential Structure of SOSS

- SOSS runs on each server node
  - Server nodes discover each other by multicast
  - Once discovered, nodes talk P2P for heartbeat and object balancing
- Cache exposed as aggregate local memory
  - SOSS assigns objects to particular nodes
  - SOSS creates replicas of objects
  - SOSS routes requests to nodes owning objects
  - SOSS caches objects at multiple points

Essential APIs

- If you leverage dictionaries and hash tables, you'll find DCP APIs simple to use
  - Objects in the cache have keys and values
- System has basic CRUD semantics:
  - Create (aka Add, Insert, Put, Store)
  - Retrieve (aka Read, Select, Get, Peek)
  - Update
  - Delete (aka Remove, Erase)

Code: Cache Client as Dictionary

```
static void Main(string[] args) {
   // Initialize object to be stored:
   SampleClass sampleObj = new SampleClass();
   sampleObj.var1 = "Hello, SOSS!";

   // Create a cache:
   SossCache cache = CacheFactory.GetCache("myCache");

   // Store object in the distributed cache:
   cache["myObj"] = sampleObj;

   // Read and update object stored in cache:
   SampleClass retrievedObj = null;
   retrievedObj = cache["myObj"] as SampleClass;
   retrievedObj.var1 = "Hello, again!";
   cache["myObj"] = retrievedObj;

   // Remove object from the cache:
   cache.Remove("myObj");
}
```

Code: Cache Client as Accessor

```
static void Main(string[] args){
    // Initialize object to be stored:
    SampleClass sampleObj = new SampleClass();
    sampleObj.var1 = "Hello, SOSS!";

    // Create a data accessor:
    CachedDataAccessor cda = new CachedDataAccessor(“mykey”);

    // Store object in ScaleOut StateServer (SOSS):
    cda.Add(sampleObj, 0, false);

    // Read and update object stored in SOSS:
    SampleClass retrievedObj = null;
    retrievedObj = (SampleClass) cda.Retrieve(true);
    retrievedObj.var1 = "Hello, again!";
    cda.Update(retrievedObj);

    // Remove object from SOSS:
    cda.Remove();
}
```

StateServer Objects

KEYS

- Uniquely identify an object within the cache
- SOSS natively uses 256-bit binary keys, but:
  - GUID = key
  - SHA-256(string) = key
  - SHA-256(*) = key
- 256-bit is an astronomically large keyspace
  - Unique value for every ~85 atoms in the universe

VALUES

- The data being stored and identified by a key
- SOSS stores values as opaque BLOBs
  - Can be of arbitrary length (but performance varies)
  - Most values are created via serialization

Design Recommendations

KEYS

- Choose meaningful keys
- Use a naming convention such as URN or URL
- Use namespaces where appropriate
- Use namespaces to differentiate usage policy

VALUES

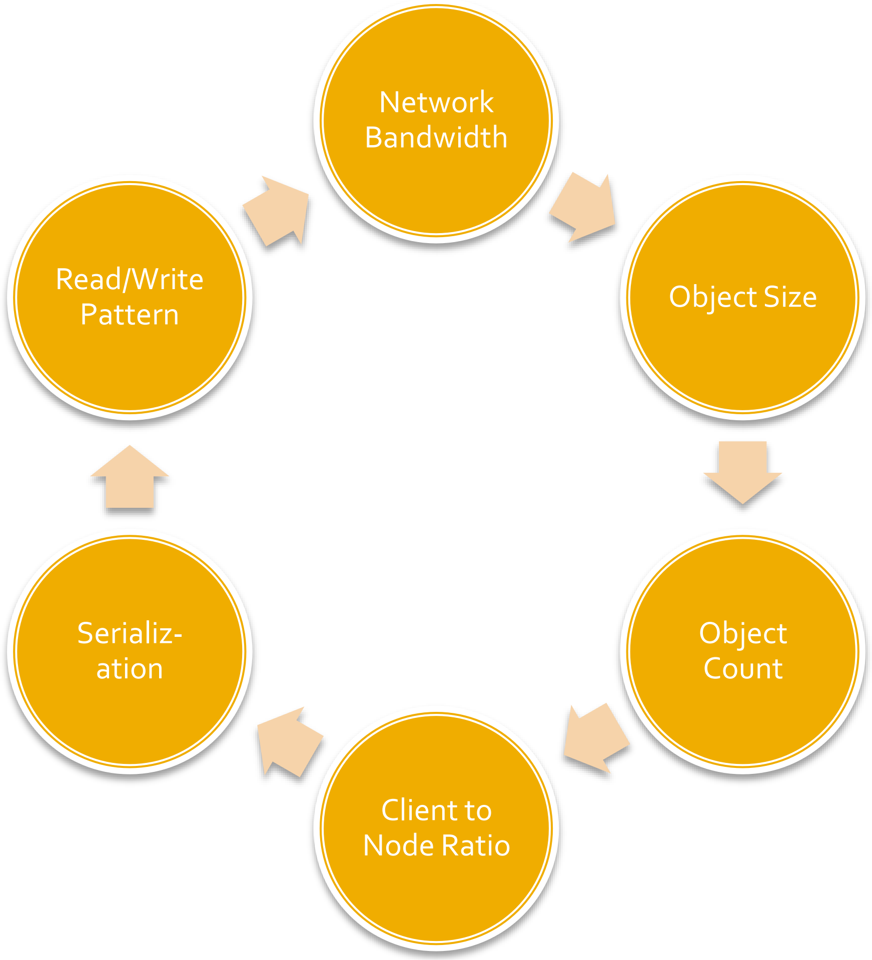
- Choose an appropriate granularity for objects
- Make read-only as many objects as possible
- Avoid designs that create hot objects

Performance

To measure distributed cache performance

Performance Variables

- Several variables affect cache performance
- They are interactive
- That said, a lot more bandwidth and nodes will go a long way

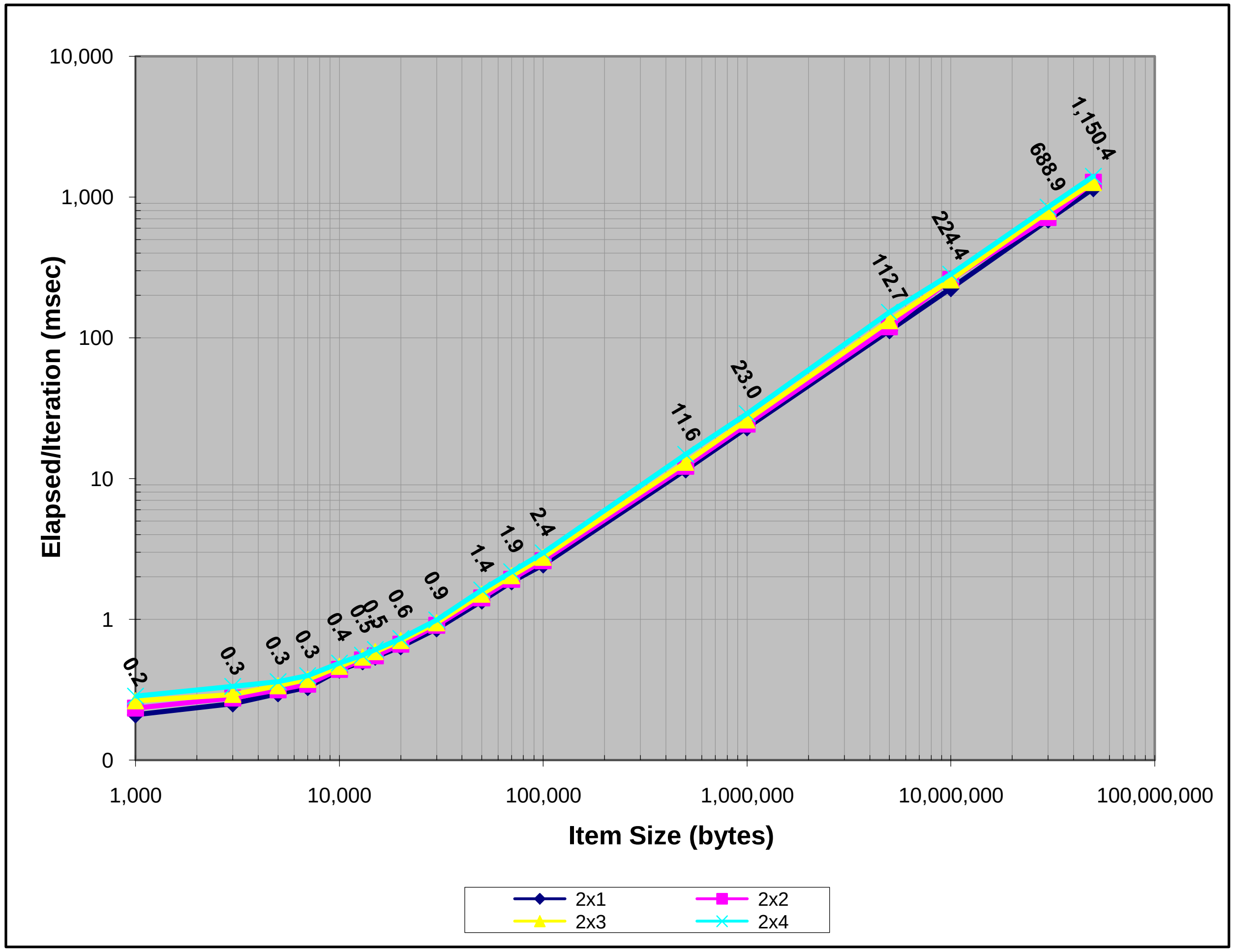


Simulation: Database Model

- This simulation reads many distinct objects from many readers simultaneously
  - Similar to many pricing workers reading information about distinct instruments
- With read caching turned off, it should behave similar to database
  - Without SQL and multirow results, of course
  - With hit on clustered index seek, rowcount = 1

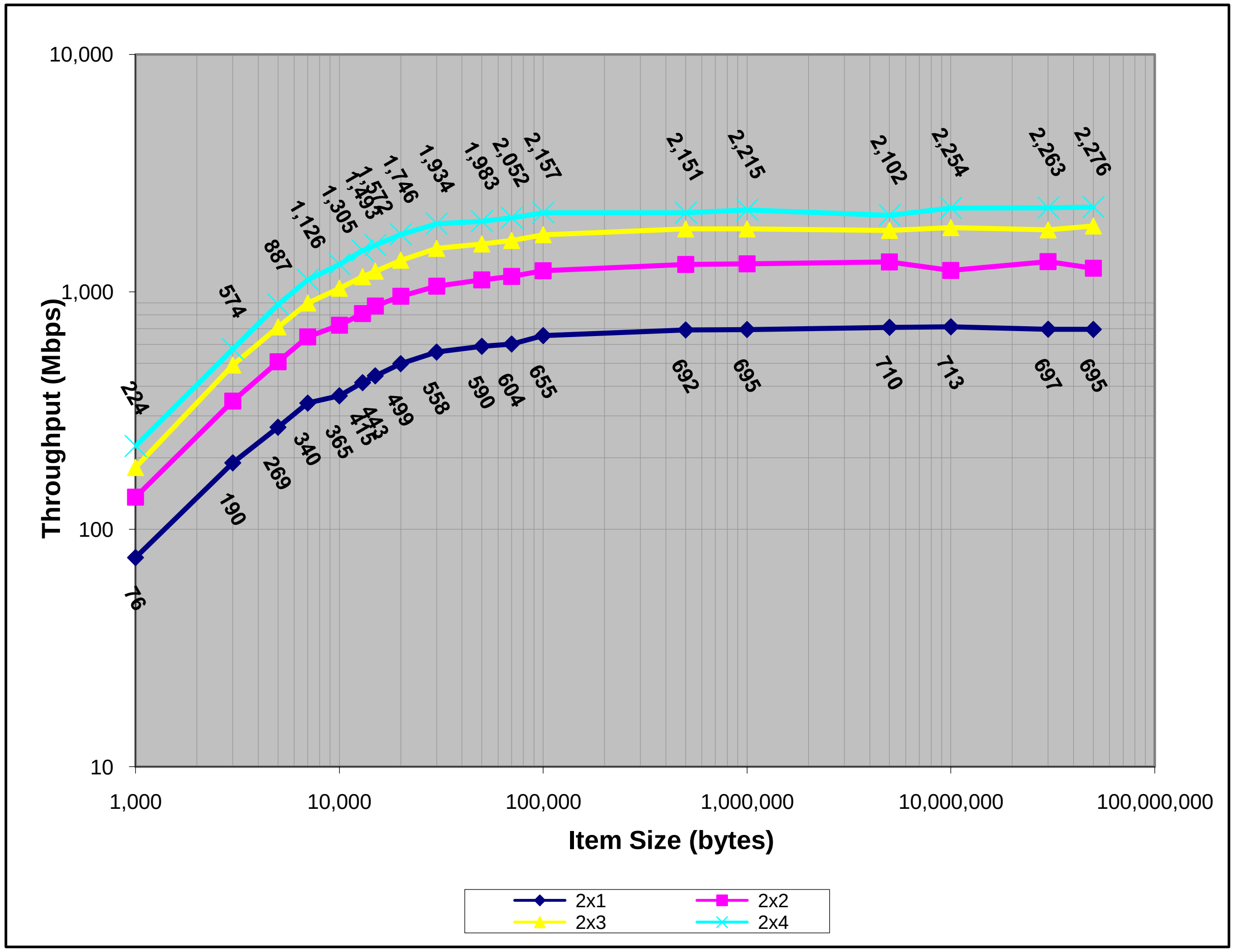
Uniform Read (*n* obj)
Cache Off
Time to Read

There is some super-linear performance on smaller objects, but it flattens out as objects get larger.



Uniform Read (*n* obj) Cache Off Throughput

Viewed as throughput, we can see that network bandwidth becomes a limiting factor.

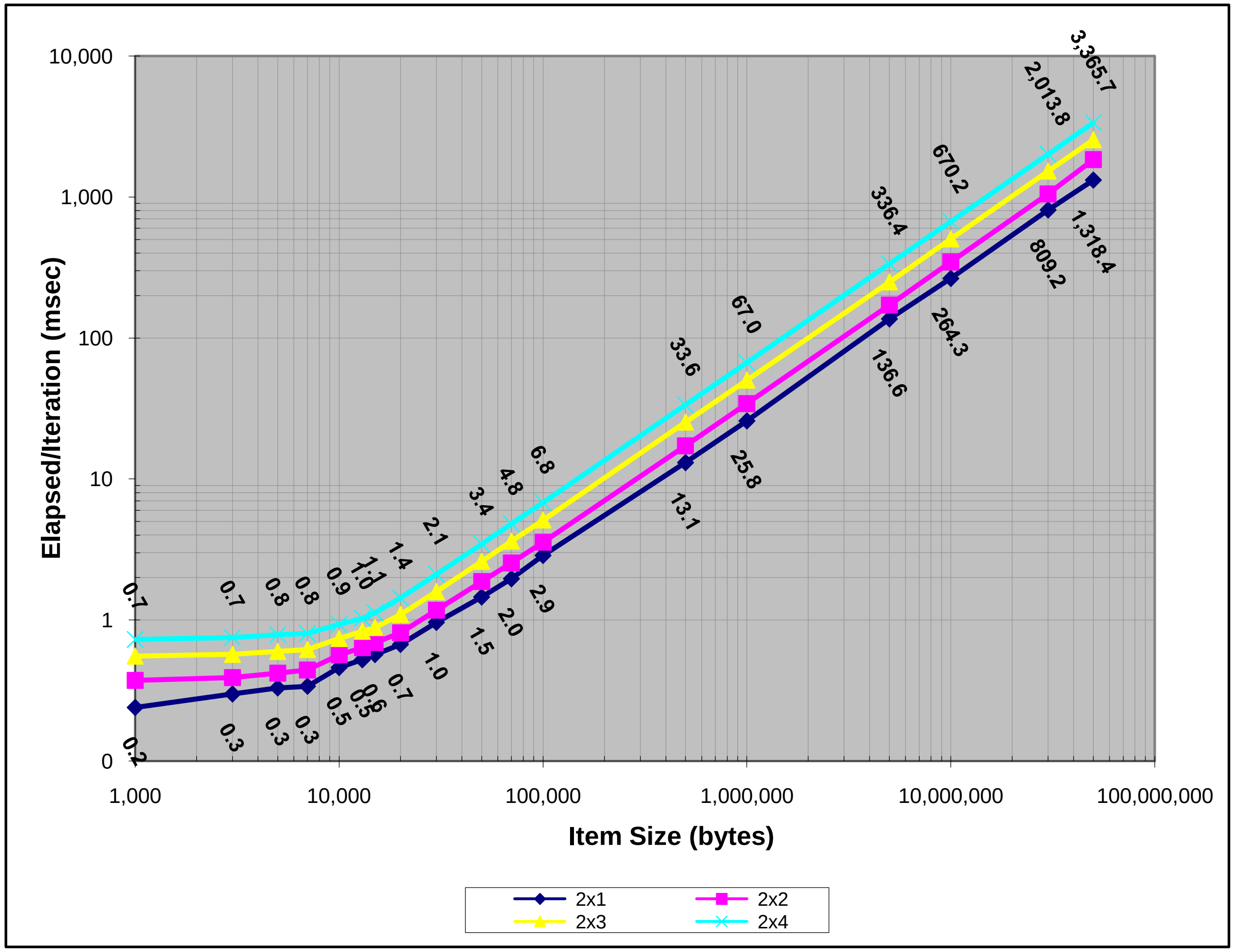


Simulation: Contention Model

- This simulation reads a single object from many readers simultaneously
  - Similar to many pricing workers reading the same common object, such as yield curve
- With read caching turned off, it should behave similar to database
  - Note that with caching off, requests are being routed to host that masters the requested object

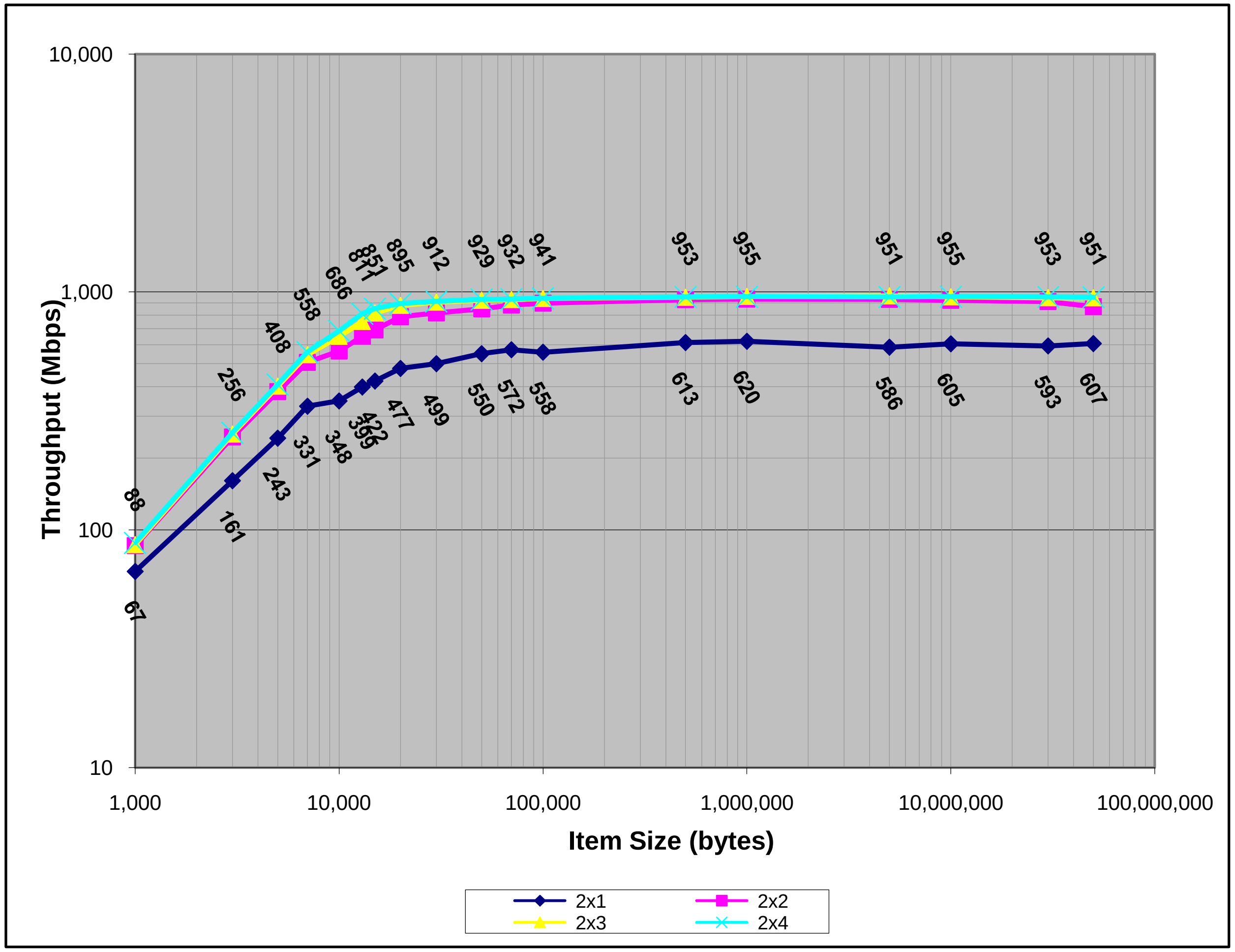
Uniform Read (1 obj)
Cache Off
Time to Read

These results are very similar to the database model, but with slightly worse performance due to contention at node that owns the master replica of the object being requested.



Uniform Read (1 obj)
Cache Off
Throughput

Again, this is performance consistent, though slower, than the previous model.

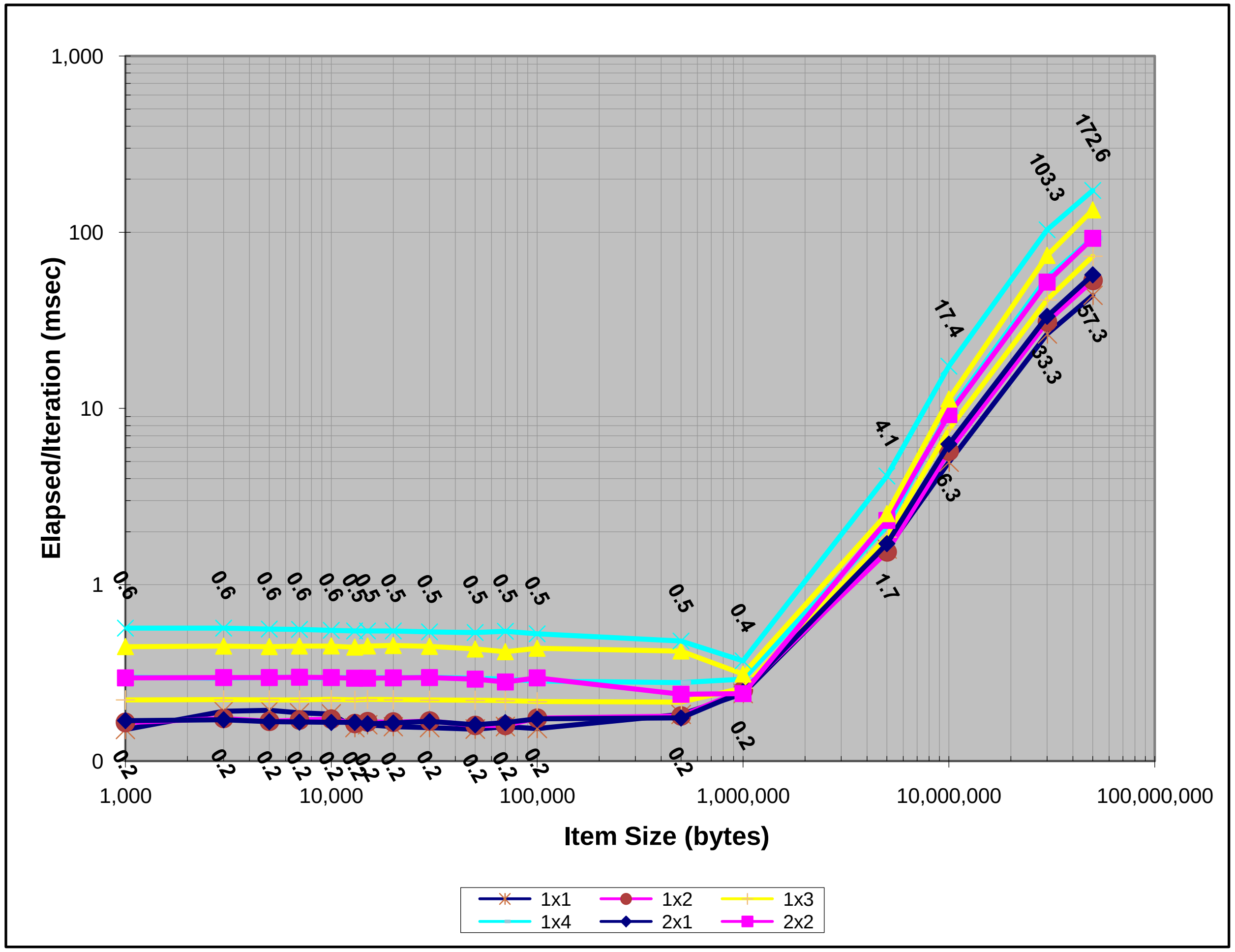


Simulation: Hotspot Model

- This simulation reads a single object from many readers simultaneously
  - Similar to many pricing workers reading the same common object, such as yield curve
- With read caching turned on, we should see much better performance
  - Note that with caching on, requests can be handled by any node that has it in cache
  - Some overhead attributed to version checking

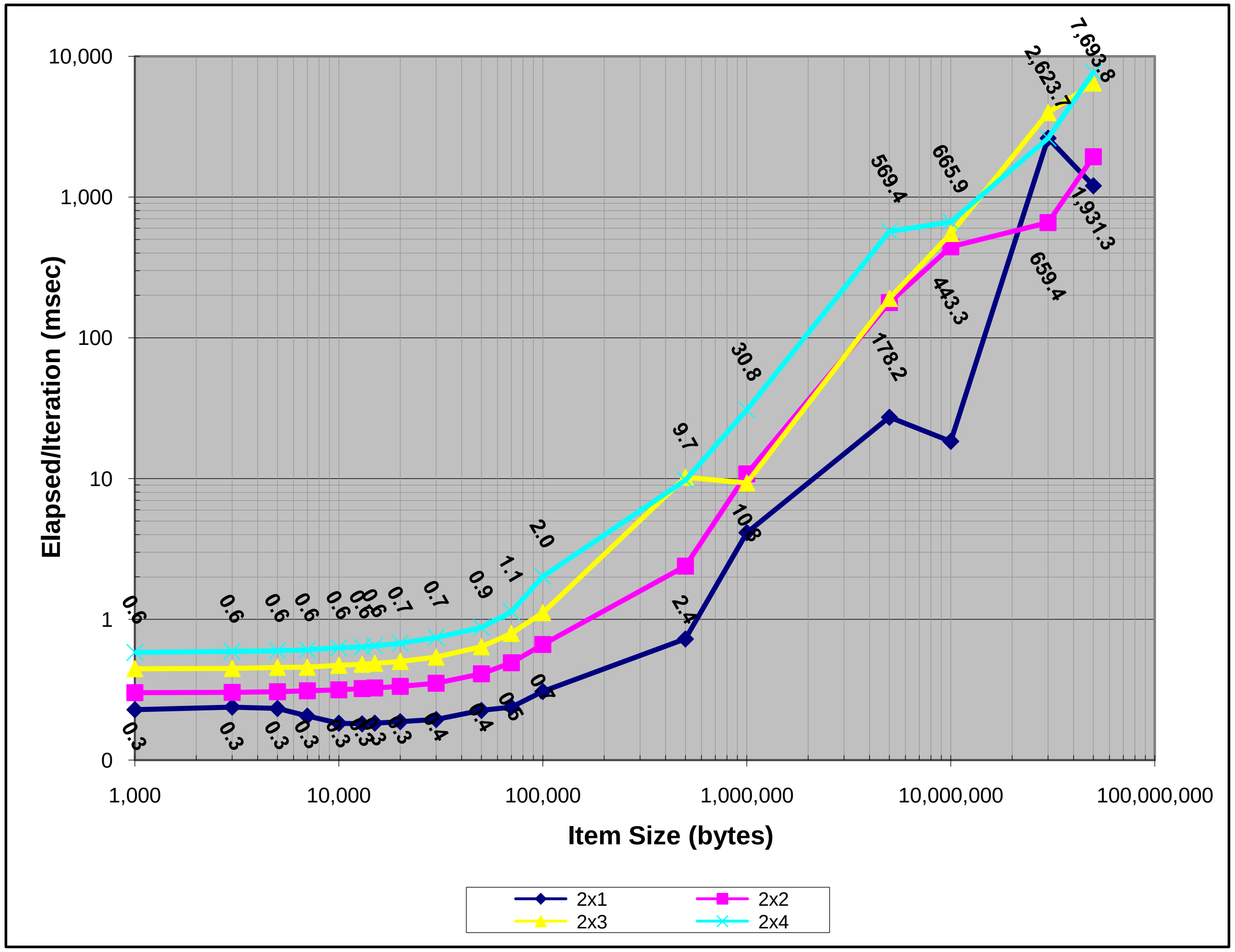
Uniform Read (1 obj) Cache On Time to Read

As expected, read times are flat up to a certain threshold as objects are returned from cache. Performance suddenly gets worse when the object sizes get large enough to saturate the network bandwidth and cause read queuing.



Uniform Read/Write Cache On Time to Read

This shows the affect of periodic updates on the hotspot object. The performance threshold is reduced due to cache invalidation.



Advanced Considerations

To squeeze further performance and functionality from caches

LAB 49
INNOVATION ON DEMAND

Microsoft Financial Services
Developer Conference

Object Compression

- The performance advantage of compression is highly contingent on object size, compression ratio, and client CPU overhead
  - For fast clients, large compressible data streams, compression can show significant gains
  - For small objects, effect can be counterproductive

Object Segmentation

- Beyond the object size threshold (usually between 100k and 200k), object segmentation can be very advantageous
  - When serialized stream is greater than threshold, object is split, pieces stored with nonce keys
  - A directory of the nonce keys is stored under the original object key
  - Safety features such as hashes/checksums push up the segmentation threshold

Cache Together or Apart?

- As DCP evolve a greater role at financial services institutions, a question usually arises:
  - Should applications own their own cache or should all share a common cache?
- Answer depends on:
  - Technology policy
  - Security requirements
  - Network and product limitations

Keys and Namespaces

- Several strategies for using string-based keys
  - URN-based locators
    - Ex: "urn:lab49-com:equity:us:msft:close;2008-03-12"
    - Ex: "urn:uuid:2af640cb-098f-4acd-9be3-9a9bd4673983"
  - Fully qualified assembly names
    - Ex: "System, Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
- Namespaces can insulate key collisions between applications or object types

Additional Resources

To find out more information about distributed caches

LAB 49
INNOVATION ON DEMAND

Microsoft Financial Services
Developer Conference

Links

- ScaleOut Software
  - http://www.scaleoutsoftware.com/
- Industry Team Blog
  - http://blogs.msdn.com/fsdpe
- Microsoft in Financial Services
  - http://www.microsoft.com/financialservices
- Technology Community for Financial Services
  - http://www.financialdevelopers.com
  - Sign up to receive the free quarterly FS Developer Newsletter on top left hand side of site

Contact

- Marc Jacobs, Director
  - mail: marcja@lab49.com
  - blog: http://marcja.wordpress.com
- Lab49, Inc.
  - mail: info@lab49.com
  - site: http://www.lab49.com
  - blog: http://blog.lab49.com