Evolving the Key/Value Programming Model to a Higher Level

Billy Newport (@billynewport)
IBM Distinguished Engineer
IBM WebSphere eXtreme Scale Chief Architect

Redis

- Redis is a pretty cool KV (key/value) store available from googlecode.
  - http://code.google.com/p/redis/
- BSD licensed code.
- It is a single process disk based store which exposes an evolved KV API.
- Does not support transactions or ACID, all disk writes are asynchronous so it's very fast.
- Relies on thrift based client libraries for sharding support, nothing built in for sharding.
- It comes with a twitter clone written in php called retwis which serves as an excellent introduction.

Redis API Basics

- Very nice API to get started with.
- The API supports the usual KV operations:
  - Get(K)
  - set(K,V)
  - Remove(K,V)
  - V Incr(K) // increment the value for the key

- But, it also supports higher level set and list operations as values for keys in a first class way.

Redis Evolved Key/Value APIs

Set APIs

- Sadd(k,V)
- Sremove(K,V)

- List<V> Smembers(K)
- Boolean SisMember(K,V)

- List<V> Sinter(K1,K2)
- N Scard(K)

List APIs

- Lpush(K,V) & Rpush(k,V)
- V Lpop(K) & V Rpop(K)

- List<V> Lrange(K,low,high)

- Ltrim(K, n)
- Rtrim(K,n)

- int Lcard(K)

List/Set operations

- First class list/set support turns out to be a big improvement on traditional be a Map KV programming.
- It simplifies many tasks involving collections of things and the developers job is much easier as a result.

- Maybe a little too easy as we'll see ☺

List operations

- For i in [0..9]
  Rpush("Members", i)

  Members -> [0,1,2,3,4,5,6,7,8,9]

- Lpop("Members") -> 0
  Members -> [1,2,3,4,5,6,7,8,9]

- Rpop("Members") -> 9
  Members -> [1,2,3,4,5,6,7,8]

- Ltrim("Members", 5)
  Members -> [1,2,3,4,5]

Redis versus conventional KV

- Redis encourages a column oriented style of programming data storage.
  - No transactions
  - No ACID
- Most DataGrids encourage an entity oriented style:
  - Transactional
  - ACID
  - A Map usually is a business object or database table.
  - Constrained Tree Schemas are typical.

Entity oriented approach

- Person p = new Person(“bnewport”, “Billy”, “Newport”, “AD34erF”)
- personMap.put(“123”, p)
- uidMap.put(“bnewport”, 123)

All attributes in one POJO/Entity

Key is usually business/data related. Data stored together under a key

Different Maps For different entities

Column oriented style: Schema free

- One global map with a common key space
- Redis applications store an entity using attributes:
  - R.set(“U:123:firstname”, “Billy”)
  - R.set(“U:123:surname”, “Newport”)
  - R.set(“U:123:password”, “AD34erF”)
  - R.set(“U:123:uid”, “bnewport”)
  - id:bnewport”, “123”)
- Each named attribute of the entity combined with the entity key becomes a key for the entries for the corresponding value.
- Space consumed is a concern though!

Map part of key

Data is split up

Column oriented style

- Awesome for prototyping or building a new system.
  - Schema free, it's all convention
  - Very easy to get started
  - Very easy to extend schema, just add columns as new keys!
  - No server side changes to extend schema, everything is just a K/V after all.
  - No transactions or anything like that.

No consistency either…

- After a while developing with this API the initial euphoria starts to wane ☺
- Biggest issue is no transactions.
  - Bugs in the application result in data issues.
- Frequent database wipes are needed because the data isn't consistent if bugs occur during development or later:
  - User billy has no password column
  - User billy wasn't added to the list of users
  - And so on.

xRedis API on IBM WXS

- Redis on IBM WebSphere eXtreme Scale = xRedis:
  - Very similar API with generics for some specific data types
  - Disk persistence provided by DB2 using purequery as API
  - Extreme Scale provides a scalable data grid which lazy pulls data from DB2 and implements write behind for high speed writes.
  - Very large caches are be readily constructed by scaling out.
- Layer on top of normal IBM WebSphere eXtreme Scale

IBM Optim Purequery

- Object oriented JDBC
- No more PreparedStatement.setXXX(int n, Object v) calls.
- Defaults assume POJO attributes have same names as DBMS columns.
- Takes SQL and list of objects and does automatic simple mapping:
  Db.updateMany("SQL", List<Person>)
- Tooling supports cases when POJO attributes have different names than columns
- Supports heterogenous statement batching if underlying DBMS supports it.
- More info @ http://www-01.ibm.com/software/data/optim/purequery-runtime/

xRedis extensions in Java

- We extended it to allow the use of a near cache.
- We support types like:
  - Long/String/Double in a first class manner
- Lists and sets are of Long/String/Double also.
- This makes the programming more type safe as well as maps efficiently to a DBMS.

Registering a new user in Java

Key Type

Value Type

```
long userid = R.str_long.incr("nextUserId");
R.c_str_long.set("un:"+username + ":id", userid);
R.c_str_str.set("u:"+Long.toString(userid)+":username", username);

String encr          word = PageUtils.hashPassword(password);
if(encrypte          == null)
    encryptedPassword = password;
R.c_str_str.set("u:"+Long.toString(userid)+":password", encryptedPassword);

R.str_long.sadd("users", userid);
R.str_long.lpush("last50users", userid);
R.str_long.ltrim("last50users", 50);

PageUtils                      userid);
```

Near Cache

Thread safe singleton for APIs

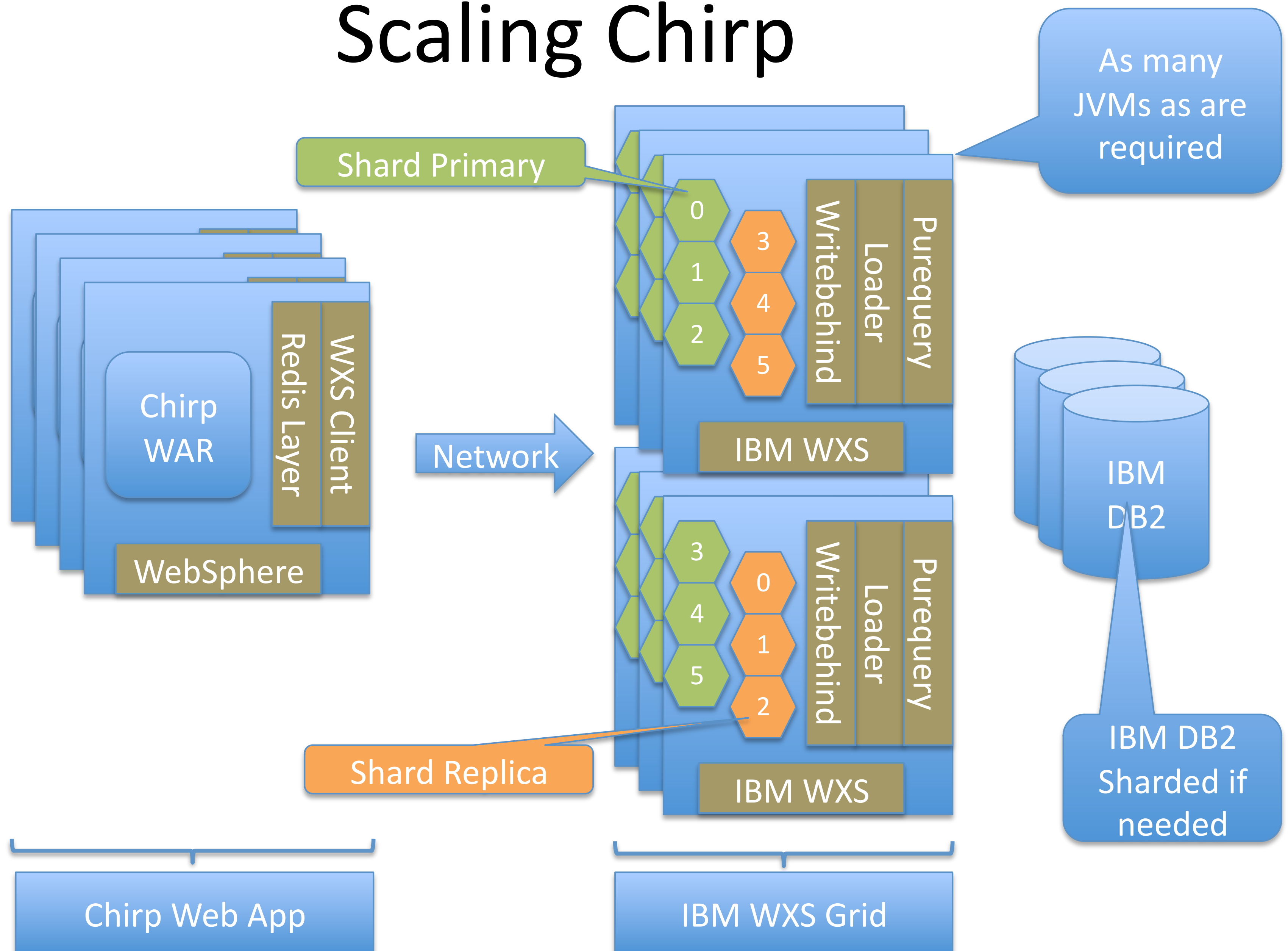
Chirp – Retwis for Java

- We ported the retwis application from php to Java using JSPs and the xRedis style API.
- The code maintains its simplicity in the process.
- Easy to understand and extend/modify.
- The redis style API is definitely a step up from KV style APIs.

Posting a new 'chirp'

```
Long postid = R.str_long.incr("nextPostId");
Long userId = PageUtils.getUserID(request);
long time = System.currentTimeMillis();
String post=Long.toString(userId)+"|"+Long.toString(time)+"|"+status;
R.c_str_str.set("p:"+Long.toString(postid), post);
List<Long> followersList = R.str_long.smembers(Long.toString(userId)+":followers");
if(followersList == null)
    followersList = new ArrayList<Long>();
HashSet<Long> followerSet = new HashSet<Long>(followersList);
followerSet.add(userId);
long replyId = PageUtils.isReply(status);
if(replyId != -1)
    followerSet.add(new Long(replyId));
for(Long i : followerSet)
    R.str_long.lpush(Long.toString(i)+":posts", postid);
// -1 uid is global timeline
String globalKey = Long.toString(-1)+":posts";
R.str_long.lpush(globalKey,postid);
R.str_long.ltrim(globalKey, 200);
%> <jsp:forward page="index.jsp"/> <%
```

Scaling Chirp



Scaling xRedis

- Redis on WXS can scale horizontally using its DataGrid capabilities.
- Better availability as process crashes don't result in data loss due to replication and more than a single process serving data.
- Each box provides more RAM/network and CPUs for redis requests.
- The list/set operations prove problematic how ever.

Large lists and sets

- The API allows programmers to add things to sets and lists and then work with them later.
- The issue is those sets/lists can be large.
- For example, Ashton Kutcher has 4 million followers.
  - Chirp/retwis stores his followers in a single set.
  - The retwis/chirp post pages fetches them and then iterates to add the new post to them.

Problems with large lists/sets

```
Long postid = R.str_long.incr("nextPostId");
Long userId = PageUtils.getUserID(request);
long time = System.currentTimeMillis();
String post=Long.toString(userId)+"|"+Long.toString(time)+"|"+status;
R.c_str_str.set("p:"+Long.toString(postid), post);
List<Long> followersList = R.str_long.smembers(Long.toString(userId)+":followers");
if(followersList == null)
    followersList = new ArrayList<Long>();
HashSet<Long> followerSet = new HashSet<Long>(followersList);
followerSet.add(userId);
long replyId = PageUtils.isReply(status);
if(replyId != -1)
    followerSet.add(new Long(replyId));
for(Long i : followerSet)
    R.str_long.lpush(Long.toString(i)+":posts", postid);
// -1 uid is global timeline
String globalKey = Long.toString(-1)+":posts";
R.str_long.lpush(globalKey,postid);
R.str_long.ltrim(globalKey, 200);
%> <jsp:forward page="index.jsp"/> <%
```

4 million
Items in this set

4 million
iterations

4 million
Server calls

Trimming lists
helps bound size

Dealing with lists

- Large lists/sets are a problem to work with synchronously
- The web page causing the operation has a very long response time.
- An asynchronous approach is needed.
- Process the large operations in chunks scheduled serially or concurrently depending on the scenario.
- Very fast RPC doesn't make up for this at all.
  - Even average chirpers have 200 or so followers…
  - There can be a lot of chirpers

Implementation of Lists

- Lists can become very large and we don't want operations on them to be proportional to the size of the list.
- Push and pop operations are constant time even if the list is currently evicted to permanent store.
- Ltrim is proportional to list length
- Lrange is proportional to the size of the range.
- Searching a list is proportional to the size of the list.

Implementation of Sets

- Sets can be partitioned in to sub sets pretty easily, use the key hash to do it, for example.
- There is no implicit order in a set so distribution is easier than lists which need range based partitioning.
- Iterating over sets then becomes a little harder as the state is now distributed.
- But, the closure can iterate over all elements in a single partition at a time for example.

Collocate or not?

- Redis is a client/server design.
- Moving all that data between the two is inefficient.
- Even if a server can do 100k RPCs/second, large lists/sets will bring it to its knees as we have seen.
- Collocating closures with the data would improve performance considerably.
- Problem is trying to keep with the simplicity of Redis which makes the API attractive:
  - No real configuration
  - No code to deploy in different places and so on.
  - Maybe groovy closures or similar

Asynchronous + Closure

- Really what's needed is:
  - Application specifies a closure which is iterated over blocks from the list or set.
  - The iteration happens asynchronously and in a guaranteed, exactly once manner.
  - Closure on data side is much faster
  - Groovy closures avoid need to distribute code between tiers in advance

- This would avoid delaying the post pages as well as be much more scalable as the system continued to grow.

Column oriented Style with a DBMS

- Great for prototyping with.
- Not so great to work with as a data source.
- No reporting, really need export utilities.
- Not easy to use in front of a 'normal' database schema.
  - Makes using off the shelf reporting tools difficult
  - Eclipse based tooling like Dali not so useful.
- But, the database schema we used is fixed and doesn't need to be customized by developer so it's easy to just setup and start.

Summary

- Redis style APIs are very interesting.
- They are great for prototyping and developing/enhancing/extending something very quickly.
- No transactions is an issue from a consistency point of view.
- The API doesn't offer scalable patterns for working with large amounts of data.
- No asynchronous invocation is an issue.
- Lack of collocation/closure support is a problem:
  - Talking about 100k gets/sec is cool but
  - If you need to talk to 4 million items, it's still a long time…
  - Working smarter is sometimes better…
  - Groovy closures look attractive for prototype

Summary

- Definitely opens up opportunities for enhancing Map style APIs moving forward.
- First class list and set support are great ideas.

- We are continuing to experiment with this API direction and are making our work so far available publicly shortly as a sample that runs on top of WXS.

More Resources – WebSphere eXtreme Scale Community

developerWorks. **http://www.ibm.com/developerworks/spaces/xtp**

developerWorks.

developerWorks > My developerWorks > Spaces >

WebSphere Extreme Transaction Processing for Developers

View the non-JavaScript version

Welcome guest | Sign

WebSphere Extreme Transaction Processing for Developers

Welcome!

Extreme Transaction Processing (XTP) Community

With the rapid growth in adoption and application of flexible and scalable ultra-high performance technology solutions, this community is our fast path to communicating directly with solution architects and developers alike.

The WebSphere Extreme Transaction Processing for Developers Space will discuss various topics for developing and deploying XTP applications and will point out emerging trends, benefits, challenges, and features associated with it.

You'll find links to our Forum to ask questions, XTP wiki, news, events, product, technical articles, tutorials and other information.

Interested in learning about other emerging WebSphere trends? Visit our WebSphere Developer's Community for Emerging Technologies to learn more.

XTP in the world

SOA and eXtreme Transaction Processing

Financial institutions are pushing the envelope and require more processing capability.

Across financial services firms we have been seeing a new set of business priorities. There are the "grow the business" priorities that are primarily centered around things like improving customer intimacy and increasing competitive differentiation. here are also ongoing issues of compliance to regulation and risk mitigation while also keeping an eye towards improving cost efficiency. The thing that hasn't changed is that IT is viewed as the enabler to overcome these challenges.

Financial institutions are pushing the envelope and require more processing capability, but without requiring exponential increase in hardware costs. The growth of extreme transaction processing (XTP) in areas such as fraud detection, risk computation, and stock trade resolution are pushing current solutions such as those based on the mainframe to the limit. These new applications require a new computing paradigm....Read More..

XTP explained

Going to Extremes: Extreme Transaction Processing

Extreme Transaction Processing - What & Why

Extreme Transaction Processing (XTP) is an exceptionally demanding form of transaction processing. Transactions of most high-end (more than 10,000 concurrent accesses or 500 transactions per second) or ultra-high-end (more than 100,000 concurrent accesses or 5,000 transactions per second) requirements or more would require this form of processing.

Gartner defines XTP as an application style aimed at supporting the design, development, deployment, management and maintenance of distributed TP applications characterized by exceptionally demanding performance, scalability, availability, security, manageability and dependability requirements.

Very much like traditional TP systems, XTP applications are aimed at enabling efficient, reliable concurrent and real-time access (read/update) to a shared database by executing application programs commonly referred to as "transactions." ... Read more.

Read the Blog here!

Explore the XTP blog which is a diary, a daily pulpit, a collaborative space, a political soapbox, a breaking-news outlet, and memos to the world on Extreme Transaction Processing.

Turbo Charge Your Applications with XTP

/dev/websphere

Twitter with us.

Are you on Twitter too? Follow us on Twitter for quick updates from WebSphere Extreme Transaction Processing Community.

Twitter with WebsphereXTP

developerWorks References

Introducing My developerWorks

My developerWorks is a new professional network and knowledge base connecting the technical communit...

Know your WebSphere Application Server options for a large cache implementation

Caching large amounts of application data doesn't always mandate the use of a 64-bit JDK in orde...

Watch us in Action!

View our XTP YouTube Channel- These are weekly video podcasts covering customers questions and forum posts on the IBM WebSphere eXtreme Scale product.

- Video Podcast: IBM WebSphere eXtreme Scale
  - Best practises for eliminating parallel queries

How to warm up JVMs correctly before measuring performance

Handling write behind errors correctly in applications

Reverse indexes, what are they and how to use them

IBM WebSphere eXtreme Scale Sessions at IBM Impact 2009

xcSet(x_10437', 'xc');

Featured Video

Best practices for eliminating parallel queries

Billy and Rob discuss how to eliminate parallel queries as much as possible for throughput scenarios .

Best practises for eliminating parallel queries

New XTP Documentation

Want some more content? Our Technical Evangelist dive deep and share articles, presentations, papers ...etc. Feel free to read, explore, and make comments and/or suggestions.

XTP Articles and Documentation

New XTP Article

Leveraging WebSphere Extreme Scale as an in-line database buffer

In this paper, we will illustrate how to optimize the performance of an application by leveraging WebSphere? eXtreme Scale as the intermediary between the database and the application.

This article will provide an overview of the theory and implementation of the write-behind caching solution and JPA loader concepts. We will then review an example business case coupled with sample code to demonstrate how to deploy these features.

PROVIDES DEVELOPERS WITH

Direct access to our technical evangelists and SMEs with channels such as:

- Blogs
- Twitter
- Forum
- YouTube Channel

Prescriptive guidance for top scenarios.

Find latest news and collateral such as articles, sample code, tutorials and demos.

© 2009 IBM Corporation

Thanks

- Please rate the session on your way out
- Please come see my other session
  “Challenges for elastic scaling in cloud environments”
  aka
  “how cloud computing is forcing middleware to evolve or die!”
- Room: “Store SAL” 14:45 on Wednesday