Value-Based Caching Framework for Data-Driven Distributed Systems

Bhagyashree Prabhakar
prabhaka@cse.unl.edu
Graduate Student
University of Nebraska, Lincoln

Data-Driven Distributed Systems

- Large and fairly static data set.
- Data set updated at known time intervals.
- Examples
  - Geospatial distributed systems
  - Digitized sky survey applications
- Same input request produces same result until data set is updated.
- Opportunity for caching

NADSS: A data driven distributed system

- National Agricultural Decision Support System NADSS (http://nadss.unl.edu)
- Geo-Spatial decision support system
- Receives requests for computing various climatic indices to assess crop production risks
- Requests span large spatial extents and long time periods
- Lengthy computations on historical climatic data slows system
- Caching can improve system response

Caching Frameworks

- JBoss Cache
  - ✓ Replicated and transactional cache
  - ✓ Cluster data across a grid of JBoss servers
  - ✓ Optimized database access
- Jakarta: Java Caching System (JCS) & JCACHE
  - ✓ Cache objects in memory
  - ✓ Lateral distribution of elements
- Oracle AS Java object cache
  - ✓ Manages local copies of objects that are expensive to create or retrieve
- Open Symphony OSCache
  - ✓ Cache JSP content, servlet responses and objects
- Tangosol's coherence
  - ✓ Clustered cache stores transient application data
- JGroups ReplicatedHashtable and DistributedHashtable

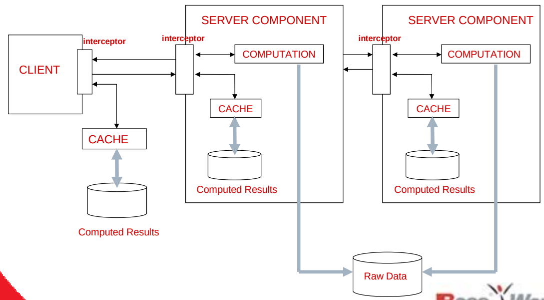
Caching in Data-Driven Distributed Systems

- Characteristics of data-driven systems influencing caching mechanism
  - ✓ Most time spent on computation
  - ✓ Repeated requests
  - ✓ Overlap in requests
- Value-Caching appropriate for data-driven systems
  - ✓ Primary goal is to save on computation time of repeated or overlapped requests
- Examine parameters to remote methods to utilize partial cache hits

Value-Caching

- Cache results of computations performed in remote methods
- Clients can benefit from earlier calls by other clients sharing the cache
- Server-side caching
  - ✓ Save on computation time
- Client-side caching
  - ✓ Save on computation and network call time
- Examine parameters and utilize partial cache hits, reducing problem size for the computation

Value Caching in a Distributed System



Cacheable Method

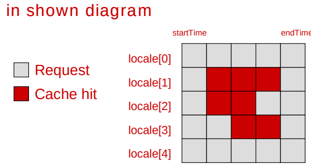
- Cache entries looked up on input parameter values
- Some parameters designated "distinct"
- Components of "distinct" parameters and hash of all other parameters makes up key for cache lookup
- Final result is a collection of intermediate results – per component of the "distinct" parameters
- Enables partial cache hits based on input parameters

1D Cacheable Method

- Example
  `int[] methodA(int[] distinctInput, String otherArgument1, int otherArgument2);`
- Result of `methodA` is a collection of results, one per each value of the `distinctInput`
- Instance:
  Client A
  ```
  distinctInput = {1,2,3,4,5};
  methodA(distinctInput,"multiply",10);
  Result = {10,20,30,40,50}
  ```
  Client B and 'n' hours later
  ```
  distinctInput = {3,4,5,6,7};
  methodA(distinctInput,"multiply",10);
  Result = {30,40,50,60,70}
  ```
- 60% partial cache hit on second call to `methodA`

2 D Cacheable Method

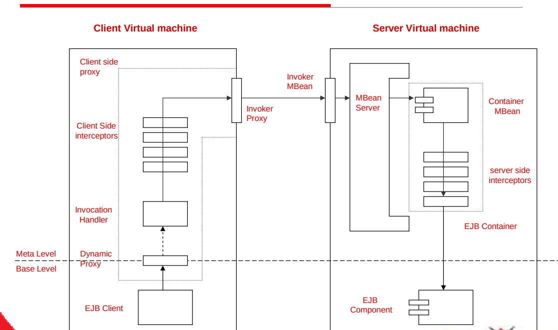
- Example
  `int[][] methodB(String[] locale, int StartTime, int EndTime, Object other Argument);`
- Result of `methodB` is a 2D array, one per each value of `locale` and each time interval between `startTime` and `endTime`
- 28% cache hit in shown diagram



Transparent Value-Caching Framework

- Non-transparent implementation of value-caching
  - ✓ straightforward
  - ✓ requires repetitive implementation for every application
- Problem generic enough to construct a transparent value-caching framework
  - ✓ can minimize or eliminate repetitive implementation
- Framework can function with little support from the application
- Framework intercepts remote method calls and delegates call to appropriate cache manager
- Cache managers are pluggable components to the framework

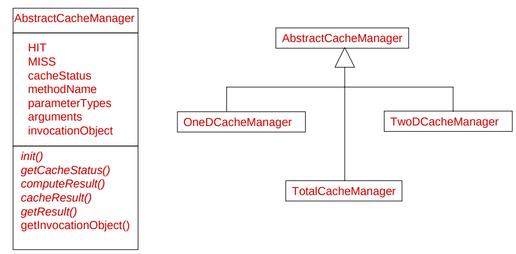
JBoss Meta-Level Architecture



Value Caching Framework in JBoss

- Results of computations in remote methods of session beans are primarily targeted for caching.
- Interceptors
  - ✓ Server side-interceptors
  - ✓ Client side-interceptors
- MBeans
  - ✓ Deploy and manage the value caching service
  - ✓ Support dynamic caching configuration for cacheable methods
- Java reflection API for data-type independence

Framework Design: Class Diagram



AbstractCacheManager

- Abstract methods that deriving managers must implement
  - ✓ ***Init()***
    - Initialize cache manager
    - Create database tables to store cached results if not present in the specified data source
  - ✓ ***getCacheStatus()***
    - Returns a boolean value indicating a complete cache HIT or MISS
    - Can maintain cache status in memory.
  - ✓ ***computeResults()***
    - Forward call to EJB on cache miss
  - ✓ ***cacheResults()***
    - Cache computed results
    - Can occur synchronously or in another thread enabling quick return of results
  - ✓ ***getResult()***
    - Return complete result
    - Complete result can be picked up from the cache after insertion
    - Result can be constructed by stitching together pieces of the result, some picked up from the cache and some computed.

Concrete Cache Managers

- Cache manager based on input parameters, computation and return type of the remote method
- Serialized result object stored in cache
- TotalCacheManager
  - ✓ Hash of all arguments is the key
- OneDCacheManager
  - ✓ Components of "distinct" parameters and hash of all other parameter values forms the cache look-up key
- TwoDCacheManager
  - ✓ 2 components of "distinct" parameters and hash of all other parameter values forms the cache look-up key

Return Type Handlers

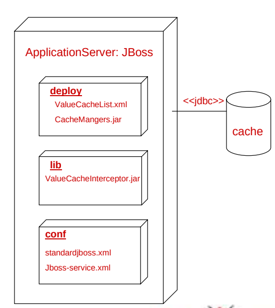
- `ReturnTypeHandler` knows how to construct the return object of the cacheable method
- Example

  `CustomReturnObject` `methodC(int[]` **`distinctInput`**`, String otherArgument1, int otherArgument2);`

  `CustomReturnObject` generally is a wrapper around a collection object.
- CacheManager is aware of the `ReturnTypeHandler`
- Value caching framework becomes more flexible and capable of handling methods not returning array objects.

Deployment

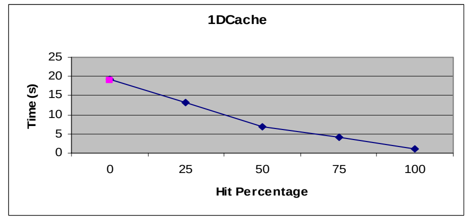
- Configuration specified in **deploy/ValueCacheList.xml**
- Value-Caching deployed as a service
- CacheListManagerMBean reads and manages value cache configuration
- MBean registered in **conf/Jboss-service.xml**
- Container configuration is modified to include Value-caching interceptor as the last interceptor



Configuration: Example

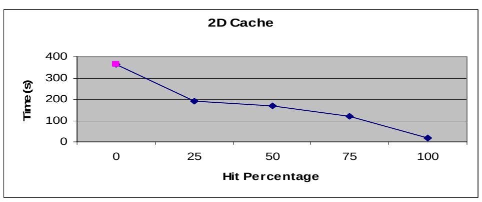
```
<ValueCacheList>
    <CacheMethod>
        <EJBClass>edu.unl.testapp.ejb.TestBean</EJBClass>
        <Method>methodA</Method>
        <Parameters>
                <Parameter distinct="yes">[I</Parameter>
                <Parameter distinct="no">
                        java.lang.Integer</Parameter>
                <Parameter distinct ="no">int</Parameter>
        </Parameters>
        <ReturnComponentType>int</ReturnComponentType>
        <CacheManager>
        edu.unl.nadss.valuecache.totalcache.OneDCacheManager
        </CacheManager>
        <CacheDataSource>CacheDS</CacheDataSource>
    </CacheMethod>
</ValueCacheList>
```


Performance: 1DCache



No Cache (s)	With Cache (s)				
	0%	25%	50%	75%	100%
18.989	19.15	13.217	6.853	4.198	1.201

Performance: 2DCache



No Cache (s)	With Cache (s)				
	0%	25%	50%	75%	100%
363.5	364.7	191.63	166.91	118.64	18.322

Observations

- Focus has been on building the transparent framework
- Performance gain depends on algorithm implemented by cache managers
- Performance of 1D and 2D Cache managers
  - ✓ Low overhead on total cache miss
  - ✓ Cache performs well even on low hit percentage
  - ✓ Write to cache after returing results to caller
  - ✓ Maintaining meta-data on cache status
  - ✓ Perform computations in chunks, such that the remote call time is amortized over computation

Future Work

- Identify frequent requests and pre-cache results
- Improve current "Best-effort" model
- Trigger automatic cache invalidation on data-set update
- Develop cache managers to cache map generation in Geographic Information Systems

Conclusion

- A transparent value caching framework is possible within JBoss
- The framework is highly configurable, and can be easily deployed as a service
- Value caching is beneficial for data driven distributed systems.

http://nadss.unl.edu

NADSS System Statistics

- Average hits per day :~50
- 100G of climatic data collected from 19,000 weather stations across the US from 1885
- SPI & PDSI are most popular index computations
- Maps generated for single year requests
- Map generation is the slowest part of retrieving results
- A Newhall report for the whole of Nebraska takes about 5 minutes to complete for a request with
  - ✓ 322 weather stations
  - ✓ 30 years from 1971-2000
  - ✓ No maps for Newhall

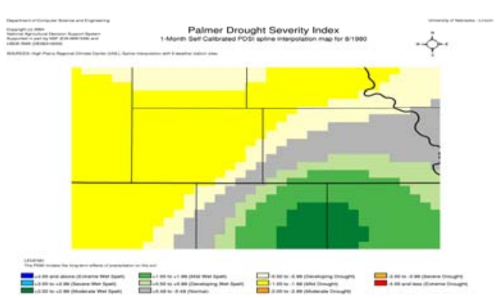
A request to the NADSS system

A Newhall request for the whole of **Nebraska**
Start Year **1971** End Year **2000**

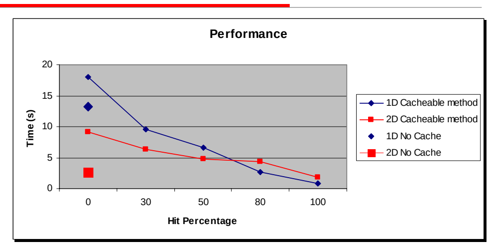
A part of the received result

Year	Prec	PET	AWB	MSD	Dry Days	MD Days	Moist Days	Bio 8	Soil Moisture Regime
1971	650.748	774.347	-123.6	-289.25	54	32	195	127	DRY TEMPUDIC
1972	790.448	746.625	43.8233	-110.5	0	21	254	239	TYPIC UDIC
1973	1490.47	764.17	726.302	-11.485	0	20	267	247	TYPIC UDIC
1974	612.14	773.536	-161.4	-231.88	55	119	131	124	TYPIC TEMPUSTIC
1975	898.652	755.455	143.197	-186.32	2	36	245	160	TYPIC UDIC
1976	637.54	751.655	-114.12	-207.75	25	153	125	142	TYPIC TEMPUSTIC
1977	1116.33	809.817	306.513	-80.845	19	26	236	114	DRY TEMPUDIC
1978	925.068	756.298	168.77	-133.67	0	16	233	221	TYPIC UDIC
1979	998.474	732.208	266.266	-28.905	0	0	253	223	TYPIC UDIC
1980	658.114	785.663	-127.55	-336.13	40	23	200	106	DRY TEMPUDIC
1981	807.466	697.915	109.551	-24.902	0	0	293	245	TYPIC UDIC
1982	956.564	705.865	250.699	-25.136	0	16	251	231	TYPIC UDIC

PDSI map for south eastern Nebraska



Performance



Cacheable Method	No Cache (s)	With Cache (s)				
		0%	30%	50%	80%	100%
1D Cacheable method	13.255	18.032	9.63	6.603	2.717	0.847
2D Cacheable method	2.666	9.127	6.313	4.745	4.355	1.798