CBOE Report 4
Focused on ETS

Jarod Jenson
Chief Systems Architect
Aeysis, Inc.
jarod@aeysis.com

Summary

The main purpose of this engagement was to focus on the end to end transactional throughput of ETS. In order to do the analysis, a full simulation was prepared to identify any potential performance inhibitors that would reduce the rate of ETS' orders per second rate.

As part of the analysis, many components of the application were monitored and there were several observations made. Specifically, it appeared that there were issues in driving ETS to its full capacity. A portion of this was due to lack of resources on non-ETS components (the systems on the FE, for instance, were not of the same hardware type), and a single thread bottleneck as well as scalability issues in ETS.

In addition, there were discussions regarding the potential use of batching for Oracle DML in the persistence server in order to increase the transaction rate by reducing network latency and unnecessary work on the part of the Oracle instance.

Detailed Analysis

One of the first items noted was there was a high number of caught exceptions occurring in the application. Analysis indicated that the thread responsible was the Talarian SmartSockets receive thread handling incoming data. It appeared, that they were using some sort of exception mechanism to deal with determining the data being received.

The exceptions themselves are not particularly onerous, however, the additional work done by the JVM to resolve the stack at the time of the exception is both a CPU intensive task as well as a scalability limiter – certain aspects of the JVM cannot change during this process. This problem was compounded by the fact that this is one of (if not the) most active thread at any given point in the applications run time. When it reaches single thread performance limitations of the CPU, the application can not processes more orders regardless of changes anywhere else in the application. It addition, the repercussions for upstream services can be severe.

Since this code is not owned by CBOE, it was decided to make use of a feature of the JVM to mitigate the issue for the sole purpose of quantifying its real effect on the application. This option is **definitely**

not recommended for production deployed code - `-XX:-StackTraceInThrowable`. Use of this would allow us to see – without code change – how much performance could be recovered if this was not happening.

The testing of this proved quite positive and netted double digit percentage increase in order throughput. A number significant enough to take this issue to the vendor (it is my understanding that this was done and that the vendor provided relief for this issue that would not require use of the debugging option).

In continuing this analysis, it is still quite apparent that this thread (even with appropriate relief from the aforementioned issue) will most likely serve as the limiting factor and most significant bottleneck to increasing order rate going forward. Options have been discussed and it is possible that the Talarian code could be removed and replaced with alternatives. It would seem quite wise to pursue this option unless the SmartSocket implementation has some performance enhancements that we are currently unaware of.

The next item noted was a lack of scalability in the application as a result of synchronization contention. As the application approached its highest volumes of throughput, there were signs of marked Java Monitor contention that, if left unaddressed, would prevent vastly superior performance even if the single-threaded nature of the above issue was made moot. Even without specific DTrace analysis, this was obvious as the application performed *better* on an 8 way system than on a 16 way system – the first and most significant sign that scalability will become an issue.

The following list is a representative set of stack frames that were shown to participate in the synchronization issues (`monitor-contended-entered`). The CBOE staff has this and other files that shown the contention points as well. For each of these that are CBOE owned components, the implementation should be reviewed, and a determination made as to whether the locking can be removed, reduced, or replaced with alternatives such as the Concurrent Libraries. Non CBOE owned frames will need to be evaluated to determine if the component usage can be reduced, changed, or eliminated.

Each line with a single integer represents the completion of the stack frame involved and serves as a relative identifier of the amount of contention.

```
              com/cboe/server/util/EventBufferHelper.startBuffer()V
              120

              com/smartsockets/TipcConnClientImpl.o()V
              136

              com/cboe/server/events/PublisherBOSessionCollectedBase.getCollector()
Lcom/cboe/server/util/ObjectCollector;
              144

              com/smartsockets/Monitor.traffic(Lcom/smartsockets/TipcMsg;ZILjava/la
ng/Object;)V
              172

              com/objectwave/persist/RDBConnectionPool.getConnection()Lcom/objectwa
ve/persist/RDBConnection;
              175
```

```
              com/cboe/instrumentationService/InstrumentorHome.findMethodInstrument
orFactory()Lcom/cboe/instrumentationService/factories/MethodInstrumentorFactory;
              356

              com/cboe/instrumentationService/transactionTiming/EventsBuffer.putInB
uffer(Lcom/cboe/idl/instrumentationService/transactionTiming/TransactionRecord;)V
              693

              com/smartsockets/TipcMsgImpl.a(Lcom/smartsockets/TipcDataOutput;)I
             1424

              com/cboe/businessServices/orderHandlingService/OrderHomeImpl.findOrde
r(Lcom/cboe/idl/cmiOrder/OrderIdStruct;)Lcom/cboe/interfaces/domain/Order;
              com/cboe/businessServices/orderHandlingService/OrderHandlingServiceIm
pl.findOrderByOrderId(Lcom/cboe/idl/cmiOrder/OrderIdStruct;)Lcom/cboe/interfaces/do
main/Order;
              com/cboe/businessServices/orderHandlingService/OrderHandlingServiceIm
pl.acceptCancel(Ljava/lang/String;Lcom/cboe/idl/cmiOrder/CancelRequestStruct;Lcom/c
boe/idl/cmiProduct/ProductKeysStruct;)V
              com/cboe/businessServices/orderHandlingService/OrderHandlingServiceIn
terceptor.acceptCancel(Ljava/lang/String;Lcom/cboe/idl/cmiOrder/CancelRequestStruct
;Lcom/cboe/idl/cmiProduct/ProductKeysStruct;)V
              com/cboe/idl/businessServices/OrderHandlingServicePOATie.acceptCancel
(Ljava/lang/String;Lcom/cboe/idl/cmiOrder/CancelRequestStruct;Lcom/cboe/idl/cmiProd
uct/ProductKeysStruct;)V
              com/cboe/idl/businessServices/OrderHandlingServicePOA._invoke(Ljava/l
ang/String;Lorg/omg/CORBA/portable/InputStream;Lorg/omg/CORBA/portable/ResponseHand
ler;)Lorg/omg/CORBA/portable/OutputStream;
              com/cboe/ORBInfra/PortableServer/Dispatcher.invokeServant(Lorg/omg/Po
rtableServer/Servant;Lcom/cboe/ORBInfra/ORB/ServerRequestImpl;)V
              com/cboe/ORBInfra/PortableServer/Dispatcher.dispatch(Lcom/cboe/ORBInf
ra/PortableServer/POAObjectId;Lcom/cboe/ORBInfra/ORB/ServerRequestImpl;)V
             2363

              com/cboe/businessServices/nbboAgentService/NBBOAgentRegistrationMapHo
meImpl.findNBBOAgentForClass(ILjava/lang/String;)Ljava/lang/String;
              com/cboe/businessServices/brokerService/BrokerStrategyFactoryStockImp
l.findNBBOAgentForClass(ILjava/lang/String;)Ljava/lang/String;
              com/cboe/businessServices/brokerService/BrokerStrategyFactoryStockImp
l.useMarketOpenStrategy(Lcom/cboe/interfaces/domain/Order;Z)Z
              com/cboe/businessServices/brokerService/BrokerStrategyFactoryStockImp
l.findOrderStrategy(Lcom/cboe/interfaces/domain/Order;Z)Lcom/cboe/businessServices/
brokerService/BrokerStrategy;
              com/cboe/businessServices/brokerService/BrokerStrategyFactory.find(Lc
om/cboe/interfaces/domain/Order;Z)Lcom/cboe/businessServices/brokerService/BrokerSt
rategy;
              com/cboe/businessServices/brokerService/BrokerImpl.acceptOrder(Lcom/c
boe/interfaces/domain/Order;)V
              com/cboe/businessServices/brokerService/BrokerServiceImpl.acceptOrder
(Lcom/cboe/interfaces/domain/Order;)V
              com/cboe/businessServices/brokerService/BrokerServiceFacade.acceptOrd
er(Lcom/cboe/interfaces/domain/Order;)V
             2485

              com/cboe/businessServices/orderHandlingService/OrderHomeImpl.create(L
com/cboe/idl/cmiOrder/OrderStruct;)Lcom/cboe/interfaces/domain/Order;
              com/cboe/businessServices/orderHandlingService/OrderHandlingServiceIm
pl.createNewOrder(Lcom/cboe/idl/cmiOrder/OrderStruct;[Lcom/cboe/idl/cmiOrder/LegOrd
erEntryStruct;Z)Lcom/cboe/interfaces/domain/Order;
```

```
              com/cboe/businessServices/orderHandlingService/OrderHandlingServiceIm
pl.acceptOrder(Lcom/cboe/idl/cmiOrder/OrderStruct;[Lcom/cboe/idl/cmiOrder/LegOrderE
ntryStruct;)Lcom/cboe/idl/cmiOrder/OrderIdStruct;
              com/cboe/businessServices/orderHandlingService/OrderHandlingServiceIm
pl.acceptOrder(Lcom/cboe/idl/cmiOrder/OrderStruct;)Lcom/cboe/idl/cmiOrder/OrderIdSt
ruct;
              com/cboe/businessServices/orderHandlingService/OrderHandlingServiceIn
terceptor.acceptOrder(Lcom/cboe/idl/cmiOrder/OrderStruct;)Lcom/cboe/idl/cmiOrder/Or
derIdStruct;
              com/cboe/idl/businessServices/OrderHandlingServicePOATie.acceptOrder(
Lcom/cboe/idl/cmiOrder/OrderStruct;)Lcom/cboe/idl/cmiOrder/OrderIdStruct;
              com/cboe/idl/businessServices/OrderHandlingServicePOA._invoke(Ljava/l
ang/String;Lorg/omg/CORBA/portable/InputStream;Lorg/omg/CORBA/portable/ResponseHand
ler;)Lorg/omg/CORBA/portable/OutputStream;
              com/cboe/ORBInfra/PortableServer/Dispatcher.invokeServant(Lorg/omg/Po
rtableServer/Servant;Lcom/cboe/ORBInfra/ORB/ServerRequestImpl;)V
             3025

              com/smartsockets/TipcConnClientImpl.p()V
             3436

              com/cboe/infrastructureServices/uuidService/IdServiceImpl.getNextUUID
()J
              com/cboe/infrastructureServices/foundationFramework/PersistentBObject
.initId()V
              com/cboe/infrastructureServices/foundationFramework/PersistentBObject
.initializeObjectIdentifier()V
              com/cboe/infrastructureServices/foundationFramework/PersistentBObject
.save()V
              com/objectwave/persist/RDBBroker.saveObjects(Ljava/util/List;Ljava/ut
il/ArrayList;II)I
              com/objectwave/persist/RDBBroker.saveObjects(Ljava/util/ArrayList;)V
              com/objectwave/persist/BrokerChangeList.doUpdates(Z)V
              com/objectwave/persist/BrokerChangeList.commitAll()V
             5182
```

The next item noted was a repeat of a prior recommendation that had not been completely implemented. The *rtserver* process (being native code) has a high memory allocation rate by many threads. The default Solaris allocator is not *MT-Hot* and therefore this significantly impacts the performance and scalability of rtserver. Simply LD_PRELOADing an alternative allocator such as `libumem(3LIB)` will avoid this issue and result in performance gains.

Another (more general) item was to evaluate the benefits of a fix created for the Sun JVM based on prior observations at CBOE (who was instrumental in getting the priority of the bug raised). This issue dealt with the relative performance of a portion of the reflection process called `isAssignableFrom`. The Sun JVM showed significant performance degradation relative to other vendors JVM's and was constantly showing up as a high CPU consumer in our profiling. With the patched version of the JVM, the relative cost of this dropped significantly and wouldn't have even been noticed if it behaved in this manner initially. It seems clear that this fix is a good and necessary fix.

These last two items are not directly issues per se, but areas that were observed or discussed. The first relates to the relative performance of the FE on two different architectures. In our testing, one system was a relatively current SPARC based system and the other a Galaxy system with AMD. It was abundantly clear that Galaxy system proved much more capable for the FE workload and ran at

approximately half the CPU utilization of the other system when it was essentially saturated. I believe that this is a clear indicator that if a 2 to 4 CPU system is to be used at this layer of the application, it will warrant a Galaxy based approach (although T2000 may be interesting, but that would need more detailed analysis).

The last major item of note that was discussed regarded using JDBC batching for the persistence server in its transactions with the Oracle database. Due to the high volume of transactions from the persistence server, a very measurable amount of time is spent merely in network round-trips as well as other overheard related to issuing single DML statements in a transaction. If the transactions could be batched, then these needless overhead could be avoided and hopefully translate into a significant increase in transactions per second.

The biggest issue to attempting this is what to do in the face of an exception. The easiest means to address this is to ensure that auto-commit is not enabled, and simply rollback all transactions in the batch when any exception occurs. Then, simply submit each transaction serially as is currently done to find the offending statement(s). The additional overhead experienced when this occurs should be minor and should almost never happen.

Secondly, in addition to having a set batch limit (say 50), there needs to be a time component as well that ensures we do not indefinitely wait for our batch limit before sending a statement. The timer would only be needed when there is more than zero un-commited transactions and it can be very brief (order of milliseconds). By doing this, latency is only incurred during a slow market and should not be noticeable (the persistence server carries some measure of latency regardless). However, the batch based approach should provide for the lowest latency in a high volume market which should be the real goal.