Oracle Complex Event Processing High Availability

An Oracle White Paper
November 2010

Oracle Complex Event Processing High Availability

Introduction....4
HA overview....4
  Purpose of HA....4
  Types of HA....5
    Active-active....5
    Active-passive....6
    Upstream backup....7
  HA quality of service....8
    Missed events....8
    Duplicate events....8
    Wrong Events....9
    Precise recovery....9
Oracle CEP HA Overview....9
  Failure Scenarios....10
  HA Adapters....12
HA Use cases....13
  HA application that publishes to external system....14
  HA design patterns....14
Adapter types....15
  Simple failover....15
  Simple failover with buffering....16
  Lightweight queue trimming....16
  Precise....17
    Connecting to external systems....17
    JMS....17
    JTA....18
    Connecting to other CEP services....19
    Coherence....20
Application considerations....20
  EPN considerations....20
    Ordering of output events....20
    Deterministic behavior....21
    Multithreading....21
    Monotonic versus nonmonotonic event ids....21

CQL considerations..22
        Application time versus system time..22
    Restart after failure..22
Benchmark Study..23
    Benchmark Methodology..25
        Load Injection..25
        Configurations Measured..25
        Metrics Collected..26
    Hardware and Software Stack..26
    Benchmark Results..27
Conclusions..28
References..29

Oracle Complex Event Processing High Availability

INTRODUCTION

Oracle Complex Event Processing (Oracle CEP) provides a modular platform for building applications based on an event-driven architecture. At the heart of the Oracle CEP platform is the Continuous Query Language (CQL) which allows applications to filter, query, and perform pattern matching operations on streams of data using a declarative, SQL-like language. Developers use CQL in conjunction with a lightweight Java programming model to write applications. Other platform modules include a feature-rich IDE, management console, clustering, distributed caching, event repository, and monitoring, to name a few.

As event-driven architecture and complex event processing have become prominent features of the enterprise computing landscape, more and more enterprises have begun to build mission-critical applications using CEP technology. Today, mission-critical CEP applications can be found in many different industries. For example, CEP technology is being used in the power industry to make utilities more efficient by allowing them to react instantaneously to changes in demand for electricity. CEP technology is being used in the credit card industry to detect potentially fraudulent transactions as they occur in real time. The list of mission-critical CEP applications continues to grow.

The use of CEP technology to build mission-critical applications has led to a need for Oracle CEP applications to be made highly available and fault-tolerant. This whitepaper describes the high availability (HA) solutions available in Oracle CEP 11g Release 1 Patch Set 2 and presents the results of a benchmark study demonstrating the performance of the Oracle CEP HA solutions. Since HA is such a complex and multi-faceted topic we first describe HA problems in general and HA problems specific to CEP. This sets the context for presenting Oracle CEP HA and gives users a solid grounding in the problem domain, so that an overall HA solution appropriate to their usage can be correctly selected.

HA OVERVIEW

Purpose of HA

Today's IT environments generate continuous streams of data for everything from monitoring financial markets and network performance, to business process execution and tracking RFID tagged assets. Oracle CEP provides a rich, declarative environment for developing event processing applications to improve the effectiveness of your business operations. Oracle CEP can process multiple event streams to detect patterns and trends in real time and provide enterprises the necessary visibility to capitalize on emerging opportunities or mitigate developing risks.

Like any computing resource CEP systems can be subject to both hardware and software faults, which, if unaddressed can lead to data- or service-loss and hence negatively impact a company's cash flow, reputation, or even legal standing.

High availability systems seek to mitigate both the likelihood and the impact of such faults through a combination of hardware, software, management, monitoring, policy, and planning. Generally HA has an associated cost and generally speaking the cost is inversely proportional to the resultant likelihood of failure. Many books have been devoted to the allocation of HA resources (for a good overview see "Blueprints for High Availability" by Marcus and Stern), but in this whitepaper we shall only consider software solutions to hardware and software faults.

Types of HA

CEP systems differ from other kinds of systems in that the data involved (events) is very dynamic, changing constantly. In a typical system, such as a database, the data is relatively static and HA systems (for example) both improve the reliability of the stored data and the availability of querying against that data. Since CEP data changes so fast storing it reliably can become problematic from a performance standpoint, or may even be pointless if the only relevant data is the latest data.

In a similar vein, CEP systems themselves are often highly stateful, building up a historically influenced view of incoming event streams, and HA must take account of this statefulness. Of course the state of the CEP system is likely to be changing as rapidly as the incoming events are arriving and so preserving this state reliably and accurately can also be quite problematic.

Typically the problem of the statefulness of the system itself is solved one of three ways; either by replicating the behavior of the system – termed active/active – or by replicating the state of the system – termed active/passive – or by saving the stream of events that produced the state so that the state can be rebuilt in the event of failure – termed upstream backup. We will now discuss these three approaches in more detail.

Active-active

As the name implies, active-active systems employ primary and secondary servers that are active. The secondary servers are also known as "hot" standbys. Active in the context of CEP means that each server is processing an identical stream of events, regardless of whether the results of that processing are actually used or not.

Figure 1 contains a high-level view of the active-active architecture. In Figure 1 each server produces an identical stream(s) of output events.

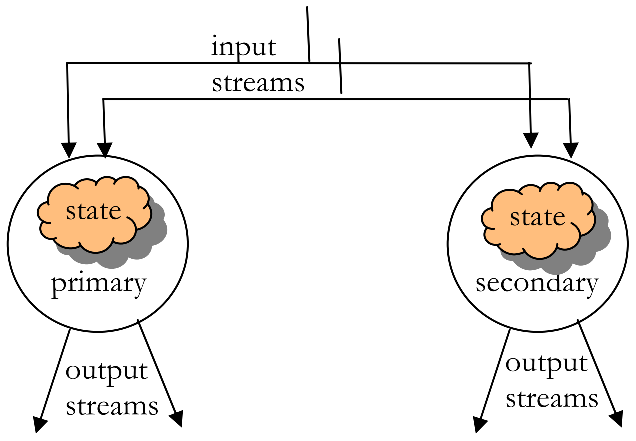

Figure 1. Active-Active server architecture.

There are two main advantages of an active-active setup – performance and simplicity. The system has good performance during normal operation and at failover time because a hot standby has little processing to do in order to take over from a failing primary. The state of the standby should reflect that of the old primary since it has processed the same set of events and the only impact at failover is actually detecting that the old primary has failed and synchronizing the new primary with the output state of the old. The system is also simple because failover does not require any state replication between servers.

In the context of CEP, which is usually highly stateful, the absence of a requirement to replicate state is very attractive. Likewise fast failover is essential in CEP systems since they are often expected to perform near real-time processing of high volume event streams. The downsides of active-active systems are that it can be difficult to build state for newly started servers, and the hardware resource requirements are high because of the redundant processing involved.

Active-passive

In active-passive systems, backups are not processing the incoming event stream; instead they are expected to take over from a failed primary through some kind of state replication. This could mean that the old primary has been spilling state to stable storage or directly to the secondary itself. Figure 2 presents a high-level view of the active-passive approach.

The advantages of active-passive systems are twofold – synchronization is implicit because state is being replicated; and resource utilization is lower than active-active since backup servers are essentially idle and could be used for additional processing.

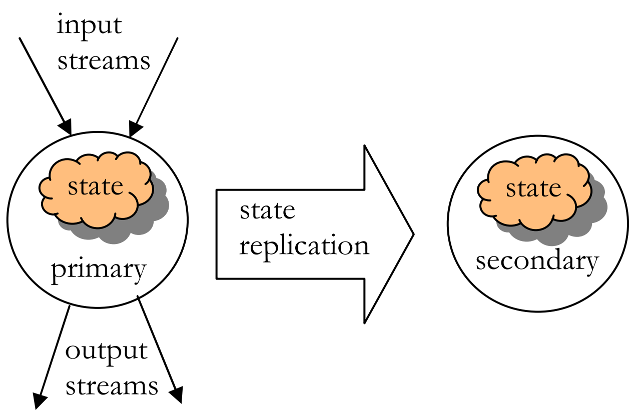

Figure 2. Active-Passive server architecture.

However, in the context of CEP active-passive systems are complicated to implement because of the need to replicate the state of the CEP system. Performance is also an issue both in terms of the demands put on the primary to replicate state; and at failover time in terms of a new primary having to rebuild its state from the replicated state.

Upstream backup

Upstream backup is a specialized case of active-passive. Instead of replicating the state of the CEP system, the incoming event stream is saved so that it can be replayed to secondaries at failover. Figure 3 presents a high-level view of the upstream backup HA architecture.

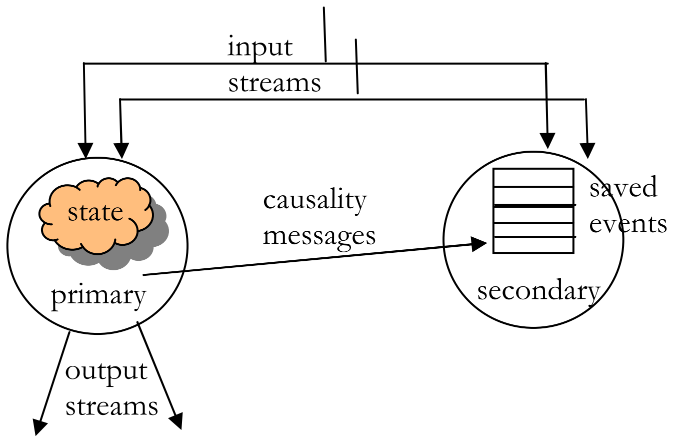

Figure 3. Upstream backup server architecture.

Saving the event stream has the advantage of not putting significant performance burden on the primary and is relatively simple to implement since an understanding of the CEP system's state is not required. The state can be thought of as being implicitly held by the saved event stream.

However, saving the incoming event stream can be costly at high event rates; and the overhead and complexity for the primary is not zero since it needs to keep a record of both where it has processed to in the event stream and which events need to be processed in order to affect the output (event causality). In degenerate situations all previously seen events need to be replayed in order to give accurate output and upstream backup is not feasible in these situations.

HA quality of service

So far we have discussed in general terms the basic mechanisms that can be employed to ensure continuity of service in a CEP system. However, as with any fault tolerant system, the details of *what* exactly happens at failover dictates the level of fidelity that can be provided. Different failover mechanisms can result in slightly different – with different levels of accuracy - results depending on the end user requirements and constraints of cost, performance, and correctness.

We can categorize the possible inaccuracies under four headings and we will discuss each in more detail below. In each case it is assumed that there is more than one server than can possibly process an input event or output a processed event. This is most common for active-active scenarios but is also possible in the active-passive and upstream backup cases since the handoff between servers may involve some loss of availability.

Missed events

Generally the most important thing that CEP users are interested in is not missing events. This covers input events – for instance it would be bad if a trading system missed or mispriced an order during failover; and output events – for instance it would be bad if an emergency services system failed to issue an alert when it had received notification of an individual entering a hazardous area.

Missed events are easy to avoid through the use of the types of redundant system that we have already described. In fact the easiest solution is to use fully redundant systems that function identically. In this instance events will never be missed (except perhaps through the loss of a datacenter) but erring on the side of caution raises another potential issue – that of output events being emitted more than once.

Duplicate events

As we have described the easiest solution to avoiding missing events is redundancy, but this raises the possibility of duplicate events. Duplicate events can also be very bad in certain circumstances – for instance it would be bad if a trading system processed a trade twice, or a banking system actioned a transfer twice! In fully redundant systems duplicate events are the norm and must be dealt with unless the receiver of events can cope. In fact being able to deal with the case where pretty much all events are duplicated also generally solves the case where only one or two are duplicated in exceptional situations – so often it is easier to design the system with this in mind. Duplicate elimination usually takes the form of first of all detecting that an event is a duplicate, and if so preventing its output. Generally this involves a computational cost and the fewer duplicates the system can tolerate the higher the cost.

Wrong Events

So far we have described scenarios that assume the incoming and outgoing stream of events is largely identical for all servers. This does not have to be the case, however. Setting aside byzantine failures caused by cosmic rays and other esoteric conditions, there still remains the largely common case of servers starting at different times. If servers start at different times then it is likely that the later one will receive a subset of the events received by the first one. This condition is even more likely when a failed server is restarted – often the restart will be needed while the system is active. On the face of it, not receiving some events doesn’t seem so bad, but problems can occur because of CEP’s stateful nature. Often events that are output are the product of a complex set of state transitions triggered by a number of previously seen events. Thus missed input events can actually lead to output events that are wrong rather than merely missing.

Precise recovery

Precise recovery means that downstream client(s) see(s) exactly the same stream of events that would have been produced if no upstream failure had occurred (missed events and duplicate events are not allowed). In some systems precise recovery is required, but the challenge is to provide precise recovery without impacting performance too greatly.

ORACLE CEP HA OVERVIEW

Oracle CEP supports an active-active HA architecture. The active-active approach has the advantages of high performance, simplicity, and short failover time relative to other approaches, as was mentioned previously. An Oracle CEP application that needs to be highly available is deployed to a group composed of two or more Oracle CEP server instances running in an Oracle CEP cluster. Oracle CEP will choose one server in the group to be the active primary. The remaining servers become active secondaries. It is not possible to specify the server that will be the initial primary as it is chosen automatically.

The number of active secondaries depends, of course, on the number of servers in the group hosting the application. If the group contains n server instances then there will be n-1 secondary instances running the application. The number of secondaries in the group determines the number of concurrent server failures that the application can handle safely. A server failure may be due to either a software or hardware failure which effectively causes termination of the server process. Note that most applications require just one or possibly two secondaries to ensure the required level of availability. Figure 4 shows a high-level view of an Oracle CEP application deployed to a group of three servers.

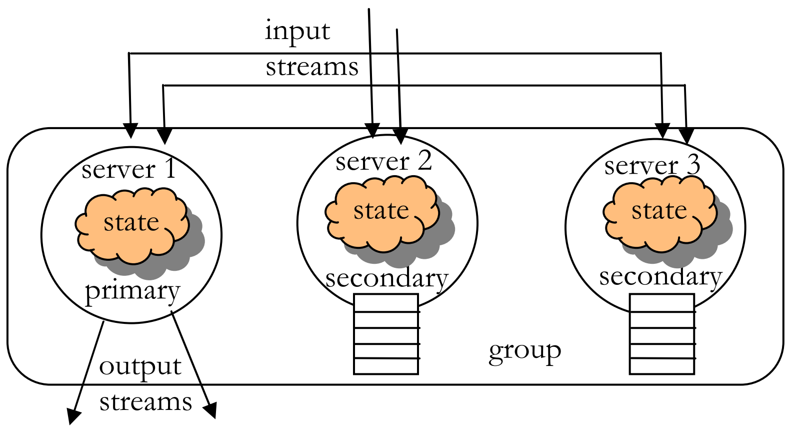

Figure 4. HA application during normal operation.

During normal operation -- prior to a failure occurring -- all server instances hosting the application process the same stream of input events. The active primary instance is responsible for sending output events to the downstream clients of the application. The active secondary instances, on the other hand, typically insert the output events that they generate into an in-memory queue. Events are buffered in the queue in the event that they are needed to recover from a failure of the active primary instance. Queued events are proactively discarded, or "trimmed", when Oracle CEP HA determines that they are no longer needed for recovery.

Failure Scenarios

Failure of an active secondary instance does not cause any change in the behavior of the remaining instances in the group, but it does mean that there is one less secondary available in case the active primary instance should fail. The active primary continues to be responsible for sending output events to downstream clients, while the remaining active secondaries continue to enqueue their output events. Figure 5 illustrates this scenario.

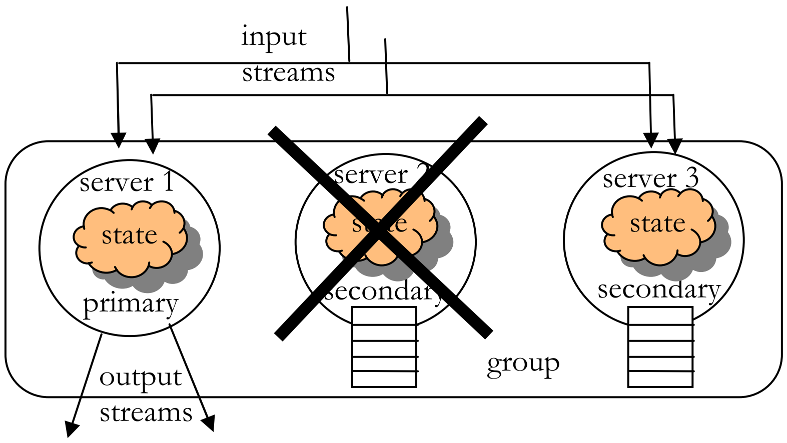

Figure 5. Failure of an active secondary instance.

Failure of the active primary instance, on the other hand, results in failover to an active secondary instance. The secondary instance becomes the new active primary and takes over the responsibility of sending output events to downstream clients. The new active primary will begin by sending the output events that are currently contained in its output queue(s) before sending any new output events that are generated following failover. Figure 6 illustrates the failure of active primary server 1. In this case, the failure has caused failover to server 3 which is now the new active primary.

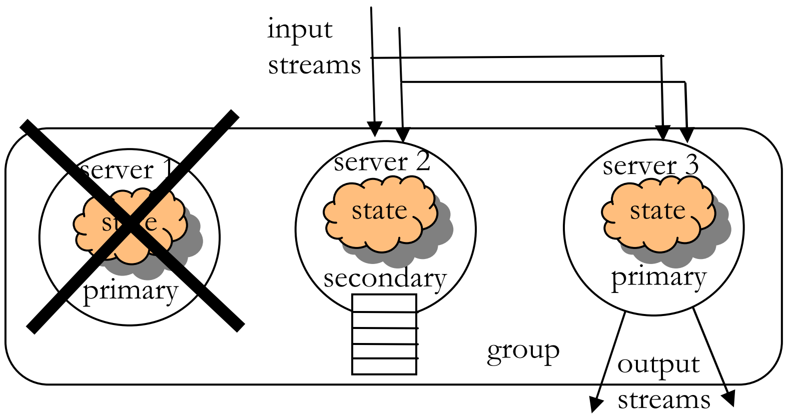

Figure 6. Failure of an active primary instance.

Multiple failures can occur as well as single failures, of course. Continuing with the example shown in Figure 6, suppose that server 3 fails after being selected as the new active primary. This results in the application state shown in Figure 7.

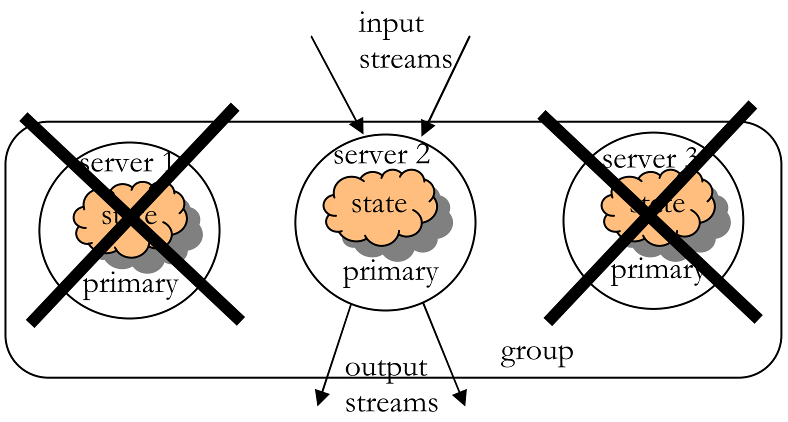

Figure 7. Multiple failures of the active primary instance.

Following the failure of active primary server 3, server 2 has been selected as the new active primary and has begun to send output events to downstream clients. From the perspective of a downstream client, the failure of server 1 and server 3 is transparent except for possibility missed or duplicate output events and a brief pause in event traffic, depending on the HA quality of service configured for the application. Since there are no additional active secondaries running following the failure of server 1 and server 3, a system administrator would need to add a new server to the group or restart a failed server before the application could safely cope with additional failures.

HA Adapters

Developers make the Oracle CEP applications that they write HA-capable by adding additional components to the application's event processing network (EPN). For a detailed discussion of the Oracle CEP programming model which includes the EPN, see the Oracle CEP IDE Developer's Guide for Eclipse which is part of the Oracle CEP 11g Release 1 (11.1.1) documentation set. The EPN components that enable HA functionality are termed "HA adapters" because there is a 1-1 correspondence between them and the regular output adapters that send the application's output events. An HA adapter can be thought of as a proxy stage in the EPN which implements HA behavior, such as queuing output events, and delegates to the regular output adapter for sending events to downstream clients.

Figure 8 shows a sample EPN that contains an input adapter which receives input events from an external system. Events flow through Channels into and out of a CQL Processor stage. Finally, the output adapter stage sends output events to downstream clients.

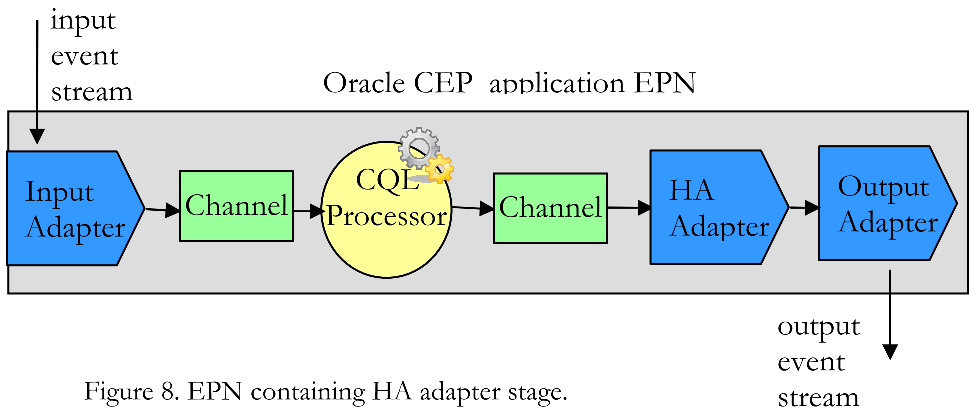

Figure 8. EPN containing HA adapter stage.

The HA Adapter acts as a proxy for the output adapter. On the active primary instance the HA adapter passes events to the output adapter so that they are sent downstream. In addition, the primary may perform other HA related processing. On the active secondary instance – remember that the same application EPN is deployed to all nodes in the group – the HA adapter typically puts events in an in-memory queue instead of sending the events to the output adapter. Oracle CEP HA provides a number of different HA adapter implementations designed to address specific application requirements. These different adapter implementations and their behavior are presented later in the paper.

HA USE CASES

"There is no such thing as a free lunch", the old adage goes and this applies equally to HA systems. Making systems more reliable involves a cost, but the kind of cost involved can be variable – it could be in terms of hardware resource or performance or accuracy, and in most HA systems customers can select different trade-offs depending on their application and operational requirements. Thus it is essential that customers understand the bounds of the systems they are implementing so that effective decisions can be made concerning HA and other operational characteristics.

Understanding this dimensionality also involves understanding the cost of failure. HA systems are typically measured in terms of "9s" of availability – thus a system with four 9's of availability would be up 99.99% of the time, or 52 minutes downtime in a year. That might not seem like a lot; but if each of those minutes costs $10m in lost revenue then it is possibly worth implementing an even more highly available system than this. If each of those minutes cost $10 however, one might wonder why four 9's is needed at all. In a similar vein, suppose spending $100m on HA is justified because of the downtime costs involved, it is pointless spending all of that money on software if the hardware is substandard, likewise it is pointless spending it all on hardware and software if there is no 24x7 operational maintenance in place, or if the electricity supply is temperamental. It is thus vital that HA be approached holistically rather than from simply a software or even technical viewpoint. Many good books on this topic exist and we would refer the reader to these for an in-depth treatise. For the purposes of this whitepaper we will assume that *all* operational concerns have been looked at, with the techniques discussed here forming a small part of the overall solution.

We will now describe the core use case that Oracle CEP HA is designed to address.

HA application that publishes to external system

An application receives input events from one or more external systems. The external systems are publish-subscribe style systems that allow multiple instances of the application to connect simultaneously and receive the same stream of messages. The application does not update any external systems in a way that would cause conflicts should multiple copies of the application run concurrently. The application sends output events to an external downstream system(s). Multiple instances of the application can connect to the downstream system simultaneously, although only one instance of the application is allowed to send messages at any one time. Within these constraints three different cases are of interest:

- The application is allowed to skip sending some output events to the downstream system when there is a failure. Duplicates are also allowed.
- The application is allowed to send duplicate events to the downstream system, but must not skip any events when there is a failure.
- The application must send exactly the same stream of messages/events to the downstream system when there is a failure, modulo a brief pause during which events may not be sent when there is a failure.

Note that in describing this use case we have treated the CEP application as a black box concerned with only input and output events. This allows us to discuss HA of the core CEP operations, but it is likely that the scope of HA for a CEP system is broader than this since the CEP application may be updating other external systems that are not event based. For instance it could be writing to a distributed cache or a database. If this is the case then careful consideration needs to be given to HA for these systems also. Alternatively the application can be structured so that these systems are essentially dealt with as event-based external systems. The key point is that it is not usually sufficient to simply improve the reliability and accuracy of event delivery – even when only considering software, the system must still be treated holistically.

HA design patterns

With this scenario in mind we can identify several design patterns that can be used to inform the HA decision-making process and improve the HA performance for a CEP application.

- Only preserve what you need. Most CEP systems are characterized by a large number of raw input events being queried to generate a smaller number of "enriched" events. In general it makes sense to only try and preserve these enriched events – both because there are fewer of them and because they are more valuable.
- Limit engine state. CEP systems allow you to query windows of events. It can be tempting to build systems using very large windows, but this increases the state that needs to be rebuilt when failure occurs. In general it is better to think of long-term state as something better kept in stable storage, such as a distributed cache or a database – since the HA facilities of these technologies can be appropriately leveraged.
- Source event identity externally. Many HA solutions require that events be correlated between different servers and to do this events need to be universally identifiable. The best way to do this is use external information – preferably a timestamp – to seed the event, rather than relying on the CEP system to provide this.
- Select the minimum HA your application can tolerate.
- Avoid coupling servers. The most performant HA for CEP systems is when servers can run without requiring coordination between them. Generally this can be achieved if there is no shared state and the downstream system can tolerate duplicates. Increasing levels of HA are targeted at increasing the fidelity of the stream of events that the downstream system sees, but this increasing fidelity comes with a performance cost.

ADAPTER TYPES

This section provides a high-level description the different types of HA adapters that are available in Oracle CEP. Developers pick an HA adapter which provides the appropriate HA guarantees for their application at design time by adding the adapter to the EPN. Different output streams in the same application can use different HA adapter types if they have different HA requirements.

Simple failover

Oracle CEP provides a callback framework which allows application instances to receive notifications when the cluster membership changes, i.e. when a server instance fails or joins the cluster. Layered upon the callback framework is a simple HA adapter which leverages the callbacks to switch on or off an outgoing stream of events. This "simple failover HA adapter" provides what might be termed "best effort" HA. More precisely, the active primary instance sends output events to downstream clients of the application, while active secondaries discard their output events. If the current active primary fails, a new active primary is chosen and begins sending output events once it is notified. Thus, output events may be missed or duplicated by the new primary depending on whether it is running ahead of or behind the old primary, respectively.

For many applications this is good enough – a temporary glitch is acceptable as long as the application is available, and accurate, for the majority of the time. Think of Yahoo!'s stock ticker for instance, transient failures may not even be seen by the majority of users – and the system provides no guarantees to end-users. Although simple failover HA cannot guarantee that output events won't be missed or duplicate events sent, it is very attractive because it has no impact on overall application performance.

Simple failover with buffering

A variant of the simple failover HA adapter has active secondaries buffer, rather than discard, events. The buffer of events can be replayed at failover to reduce the chance of missed events. This scheme, while simple and performant, has the disadvantage of outputting a significant number of duplicates at failover when larger buffers are employed. Of course larger buffers also reduce the chance of missed messages so we once again see a tradeoff in the approach. Figure 4 shows simple failover with buffering.

Lightweight queue trimming

If an application is tolerant of the occasional duplicate, but cannot tolerate missed messages then a natural extension to lightweight buffering is lightweight queue trimming. When using the queue trimming HA adapter, the active primary communicates to the secondaries the events that it has actually processed. This enables the secondaries to "trim" their buffer of output events so that it contains only those events that have not been sent by the primary at a particular point in time. This allows the secondary to avoid missing any output events when there is a failover -- since events are only trimmed after they have been sent by the current primary.

The frequency with which the active primary sends queue trimming messages to active secondaries is configurable. Queue trimming messages can be sent on an event basis – every n events (0<n) -- which limits the number of duplicate output events to at most n events at failover or on a timed basis – every n milliseconds (0 <n). The queue trimming adapter requires a way to identify events consistently among the active primary and secondaries. This is a requirement that the simple failover and buffering HA adapters do not have. The recommended approach is to use "application time" to identify events, but any key value that uniquely identifies events will do. The use of application time to identify events is discussed in more detail later in the paper.

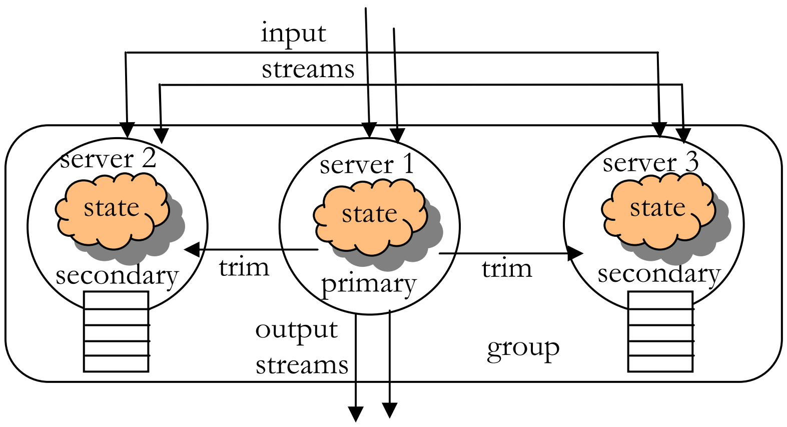

Figure 9. HA adapter using lightweight queue trimming.

The advantage of queue trimming is that output events are never lost. There is a performance overhead at the active primary, however, for sending the trimming messages that need to be communicated and this overhead increases as the fidelity of trimming increases. Fortunately the Oracle CEP container is able to leverage the high performance TCMP protocol from Oracle Coherence so that the impact of this is minimized.

Precise

If duplicates simply cannot be tolerated then a precise recovery adapter is provided by the OCEP system to output a single stream of events. The mechanism used to achieve this varies depending on the requirements and downstream system involved, but all solutions require distributed correlation of events. Typically this will be done through Coherence, which has proven mission-critical performance in this space.

Connecting to external systems

Two type of external system in particular are catered for by the precise adapter – JMS and JTA.

JMS

Oracle CEP provides a JMS adapter out-of-the-box for connecting to external systems. The majority of messaging-oriented and event-oriented products support some kind of JMS access and so the JMS adapter is a universally applicable and easy-to-use choice for connecting to external systems. JMS providers are generally highly optimized and support very high message rates thus being a good fit for CEP type applications. In terms of HA, JMS is also a very attractive technology supporting both message-level delivery guarantees and publish-subscribe style connectivity.

Typically HA in JMS involves transactionally enqueuing or dequeuing messages, but in a CEP system this can have quite a significant impact on performance and also is often simply inappropriate to the application style. For instance there may not be a 1-1 correspondence between input and output events so coordinating with the upstream system can be problematic. Fortunately in an active-active setup there is a reasonably simply solution to these problems. In active-active we are not concerned with transactional guarantees along the event path for a single-server but in guaranteeing a single output from a set of servers. To achieve this secondaries can be setup to listen, over JMS, to the event stream being published by the primary. As Figure 10 shows, this incoming event stream is essentially a source of reliable queue-trimming messages and so can be used to trim the output queue of the secondaries. If JMS is configured for reliable delivery we can be sure that the stream of events seen by the secondary is precisely the stream of events output by the primary and thus failover will allow the new primary to output precisely those events not delivered by the old primary.

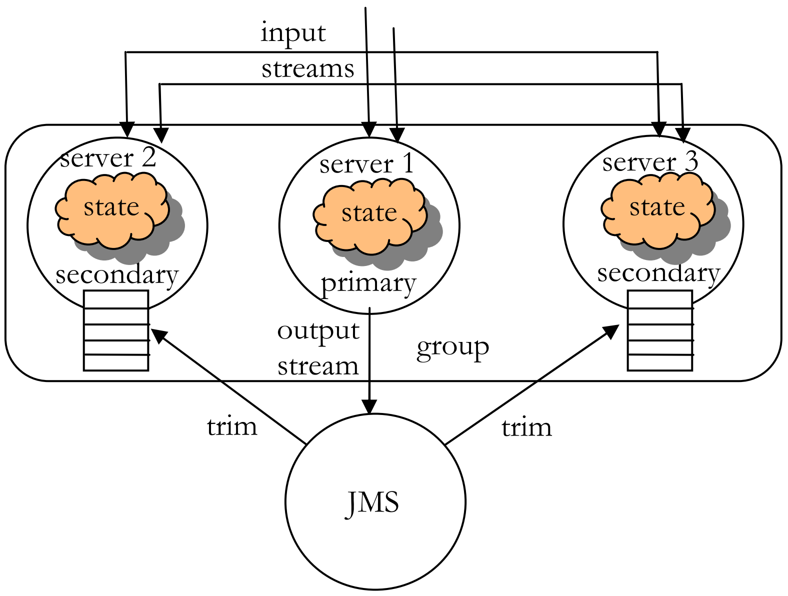

Figure 10. Precise HA adapter using JMS.

The only configuration necessity is that of making sure the new primary has indeed seen all of the events published by the old primary and a simple timeout suffices here.

JTA

In the previous section we alluded to the use of JTA in CEP systems and how this may not be the most appropriate technology in this instance. However, as we have also discussed, it is not just the CEP engine that needs to reliable; the CEP application as a whole will likely involve other external resources such as distributed caches and databases and it is the update of these resources that may well require the use of JTA.

JTA ensures the ACID coordination of updates between transactional resources. Thus, if a CEP system is publishing to a transactional downstream system – usually

JMS – and is updating a transactional store – e.g. Coherence – it makes sense that these updates be coordinated so that the store does not contain data for unpublished events or vice versa.

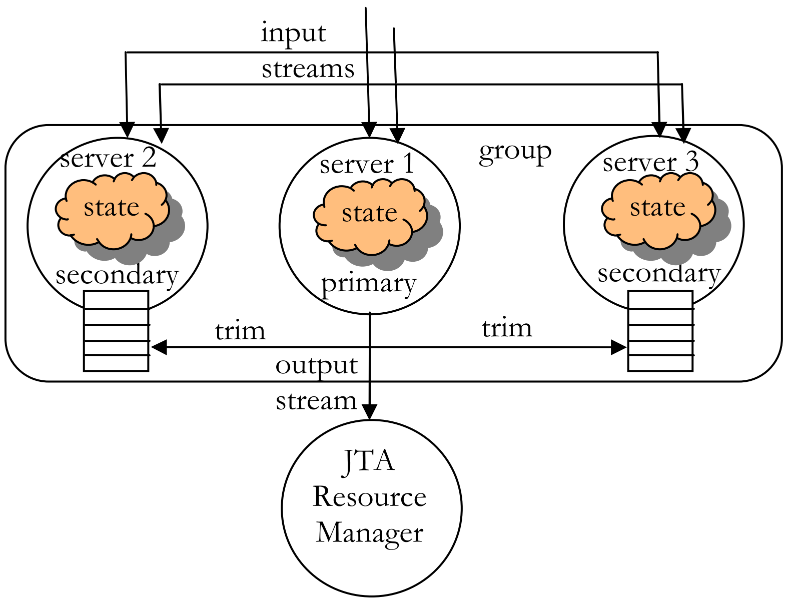

Figure 11. Precise HA adapter using JTA.

Figure 11 shows a high-level view of an application that outputs events to a downstream JTA-enabled system using the JTA adapter. Output events and queue trimming messages are sent to the downstream JTA resource manager as well as to the active secondary instances as part of the same JTA transaction. This ensures that secondaries only trim messages that have been received by the downstream client. Batching can be used to improve performance.

Connecting to other CEP services

Oracle CEP provides a variety of mechanisms for connecting to other systems and, being based on the Spring framework, supports all of the connectivity options supported by Spring. However, the vast majority of these connectivity options are not transactional and this fact needs to be constantly borne in mind when developing an HA-ready application. As with any HA system, the reliability of the system is governed by its weakest link – single points-of-failure will mean that the entire system is vulnerable to catastrophic failure, even though many pieces may be fully HA enabled.

Thus, for instance, it may be counter-productive to use precise recovery for output events if these events are correlated with updates to an unreliable webservice.

In all of these scenarios a good practice is to treat the CEP system as a black box and then consider all of the ways that information can get in and out of the system. Understanding these information flows will then enable you to make appropriate decisions about the type of HA to employ in the application. Understanding information flows will also inform exactly where to place HA capabilities – for instance it often makes sense to update external systems in the final stages of your EPN, rather than making scattered updates throughout the application. In this way, even if the external systems are not transactional, a reasonable level of data consistency can be attained.

Coherence

We have described how CEP state should be considered as either "long-term" or "short-term" so that other technologies can be employed for storing long-term data. An ideal technology for storing longer term data is Oracle Coherence. Not only does Coherence have an impressive track record in the reliable storage of in-memory data, but Oracle CEP provides direct integration with Coherence such that Coherence cache's can be treated as sources and sinks of event information. Often in CEP applications frequent updates need to be made to application state where roundtrips to the database would negatively impact performance. Since Coherence is able to handle the storage of data in-memory, much greater performance is achievable without a reduction in availability.

APPLICATION CONSIDERATIONS

This section discusses the things that application architects and developers need to bear in mind as they design and write applications that will ultimately need to be made highly available. Typically these "design considerations" only effect a subset of HA adapter types, so it is important to keep in mind the HA requirements of the particular application, and hence the HA adapter that will be used, when deciding which design considerations need to be observed. This illustrates an important fact which is that the application should be designed with HA in mind from the very beginning; rather than HA being added at the end of the development process.

EPN considerations

This section describes the constraints and best practices that should be observed when designing the EPN of an HA application.

Ordering of output events

In some cases it is important that the active primary and secondary instances generate not only the same output events, but also that they generate them in exactly the same order. This issue affects the lightweight queue trimming adapter shown in Figure 9. When using this adapter, generating output events in different orders can lead to either missed output events or unnecessary duplicate output events when there is a failure. Let's take a look at why this is so.

Suppose the application's output events have a unique key value that identifies them. Call this property of the output events the "event id". The event id could be a transaction id in a credit card application, or a claim id in an insurance fraud application, or a similar unique value in a different application domain. Figure 12 shows a partial stream of output events produced by the fictitious application.

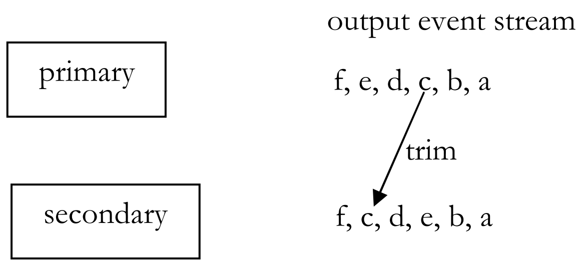

Figure 12. Effect of out-of-order events.

In Figure 12, events on the right in the stream are output first. For example, event 'a' is output first by both the primary and secondary. Now, suppose that the primary decides to send a queue trimming message after event 'c' is output. This will cause the secondary to trim all events in its queue generated prior to event 'c' including event 'c' itself. In this case the set of events trimmed will be {a, b, e, d, c} which is erroneous because events ‘d’ and ‘e’ have not yet been sent by the primary. If a failover occurs after processing the trimming message, events will be lost.

Deterministic behavior

In order for an application to generate events in the same order when run on multiple instances, it must be deterministic. This means that the application must not use things like a random number generator that may return different results on different machines. The application also must not rely on the results of methods like System.getTimeMillis() or System.nanoTime() which can return different results on different machines because the system clocks are not synchronized.

Multithreading

Multithreading is another source of nondeterministic behavior in applications since thread scheduling algorithms are very timing dependent. This can result in different threads being scheduled at different times on different machines. One should be aware of the threading model underlying the EPN. It is important, for example, to avoid creating an EPN in which multiple threads send events to the ha adapter in parallel which can lead to the problems discussed above. For example, configuring a Channel with multiple threads and making the channel an event source for the HA adapter would cause events to be sent to the adapter in parallel by different threads and could make the event order nondeterministic.

Monotonic versus nonmonotonic event ids

Using a monotonic id to identify events instead of an id that only supports equality comparisons is another best practice when using the lightweight queue trimming adapter. This adds a level of robustness to the queue trimming algorithm since secondaries can use the monotonic nature of the id to ensure that day never trim an event whose id follows an event that has been processed by the primary.

CQL CONSIDERATIONS

This discusses best practices and restrictions related to the use of CQL in HA applications.

Application time versus system time

This issue affects all HA adapters since it can cause primary and secondary instances to output different, not just differently ordered, event streams. In Oracle CEP each event is associated with a point in time at which the event occurred. There are two general flavors of time: application time and system time. Application time means that a time value is assigned to each event "externally" by the application before the event enters the CQL processor. System time, on the other hand, means that a time value is associated with an event when it arrives at the CQL processor, essentially by calling System.nanoTime().

Application time is generally the best approach for applications that need to be highly available. The application time is associated with an event before the event is sent to Oracle CEP, so it is consistent across active primary and secondary instances. System time, on the other hand, can cause application instances to generate different results since the time value associated with an event can be different on each instance due to system clocks not being synchronized.

Using system time is not a problem for applications whose CQL queries do not use time-based windows. Applications that use only event-based windows depend only on the arrival order of events rather than the arrival time, so system time may be used in this case. For applications that need to use system time and that also use time-based windows in CQL, Oracle CEP provides a special input adapter that intercepts incoming events and assigns a consistent time that spans primary and secondary instances.

Restart after failure

We have discussed how active secondary servers can provide seamless availability in the presence of failure of a primary, however in order to provide high-levels of availability it is important that failed servers can be restarted, or new secondaries brought online. Restarts of this kind present a particular issue with regard to the state of the application, in that it must be possible for the new server to “catch up” with the state of the already running servers.

This problem is actually non-trivial to solve in a general way and application developers must give it consideration, especially with respect to persisted state. The simplest solution to this problem is to wait – the assumption being that if long-term state can be read from stable storage then short-term state will be rebuilt within some time period when a sufficient number of incoming events have been seen. This is one reason why it's important not to rely too much on the query engine for storing long-term state – it makes it difficult for servers to ever catch up.

During this catch-up period it's possible that the event output of the new server will differ, perhaps significantly, from other secondaries in the group. This is because the query engine is working with incomplete state. Now although this may not matter because the secondary is not actually outputting any events, it does cause a problem with HA strategies that rely on correlation between event streams of different servers. For this reason it is generally sensible for new secondaries to not take part in HA algorithms until the catch-up window has expired.

BENCHMARK STUDY

This section describes a benchmark study that demonstrates the performance and scalability of the Oracle CEP product in various HA configurations. The application used in the benchmark implements a Signal Generation scenario in which incoming streams of data are monitored for the occurrence of a certain conditions that then trigger the generation of output events. This is a very common CEP usecase in financial services (algorithmic trading) as well as other industries. The input data for the application consists of simulated stock market data (symbol and price) for 1460 distinct stocks, with time varying prices. For each input event, the CQL processor initially applies a filtering query to determine if the symbol is on a watch list of 300 symbols being monitored. This initial filter reduces the input data by roughly a factor of 5. For events matching the 300 monitored symbols a subsequent CQL query applies a pattern match that determines whether the price for a given stock increases or decreases by more than 2 percent from the immediately previous price. Any time an increase or decrease of more than 2 percent is detected, an output event is generated. The net result is that the output rate is approximately 5 percent of the input rate over the duration of a benchmark run.

The CEP cluster configuration for the benchmark consisted of two machines, and four total CEP server instances. Each machine hosts one primary and one secondary as shown in Figure 13. For both scaling and HA purposes, the cluster is configured with 2 groups. Each group is configured with a primary and a secondary server such that the primary and secondary for a given group are on separate machines. The same application is deployed to both groups, but the input data is partitioned with half of the input delivered to group 1 (servers 1 and 3) and the other half of the input delivered to group 2 (servers 2 and 4). In this specific benchmark application the partitioning is based on stock symbol. All output data is sent to a JMS topic hosted by a separate Oracle WebLogic Server cluster. In the event of a failure of either primary server (or the

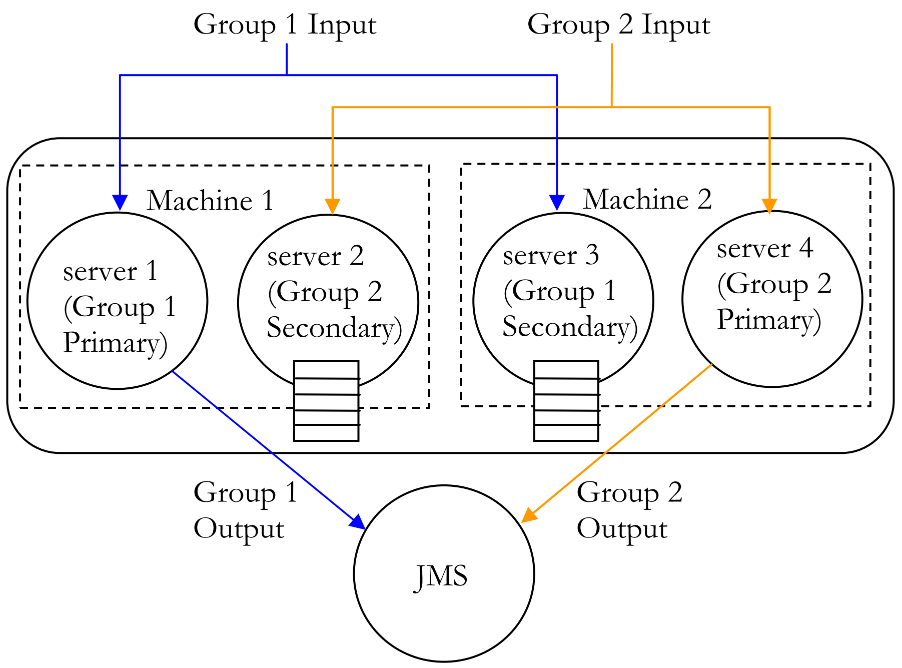
Group 1 Input
Group 2 Input
Machine 1
Machine 2
server 1 (Group 1 Primary)
server 2 (Group 2 Secondary)
server 3 (Group 1 Secondary)
server 4 (Group 2 Primary)
Group 1 Output
Group 2 Output
JMS

Figure 13. Benchmark Cluster Configuration.

machine hosting the primary) the active secondary for the group will become the new group primary and begin sending output events for that group (beginning with

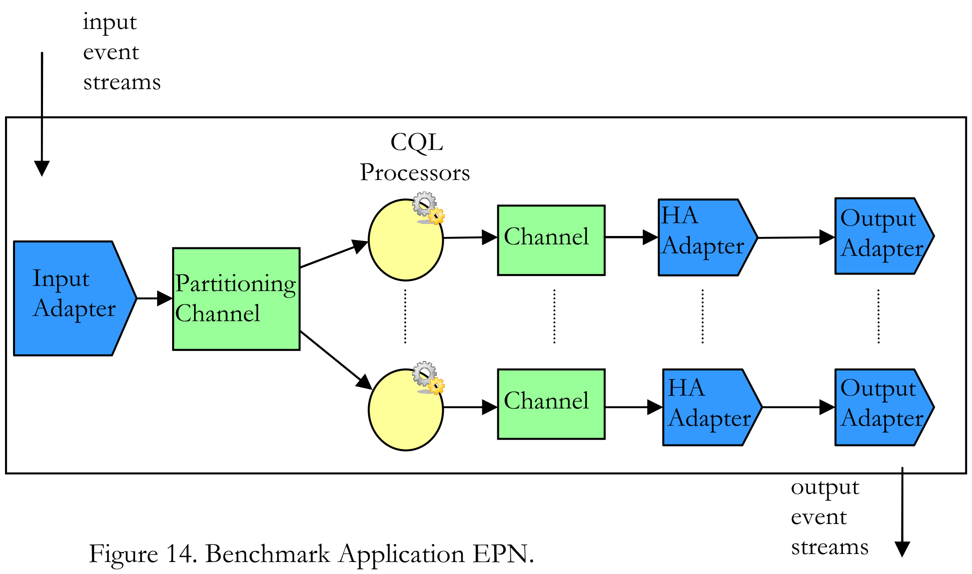

Figure 14. Benchmark Application EPN.

any queued events). This configuration allows a CEP cluster to provide both scalability and availability simultaneously and can be used with any of the HA adapters and qualities of service described earlier.

The configuration of the application's EPN within a given server instance is shown in Figure 14. Because the benchmark application does order-based comparison of stock prices, correct behavior of the application requires that all events associated with a given stock symbol be processed in order. To maintain this guarantee and minimize locking overhead, the input events for a given server instance are further partitioned within the server instance with each partition handled by a separate thread. In order to meet the determinism requirement discussed in the previous section the HA adapters and CQL processors are single threaded with separate adapter and processor instances per partition.

Benchmark Methodology

Load Injection

The input data was fed into the system by a load generator that was configured to partition the total set of stock symbols such that half of the stock symbols are sent to group 1 (servers 1 and 3) and the other half are sent to group 2 (servers 2 and 4). The input is sent at a configured, metered rate (referred to as the *injection rate*) which is identical for all servers in the cluster. Within a group, the load generator sends an identical input stream (same symbols and prices in the same order) to both servers. Ordering is based on an external timestamp applied by the load generator which ensures determinism across the servers in a group (see earlier discussion of application time vs. system time).

Each benchmark run consisted of a 10 minute warmup run, followed by a 10 minute measurement run at the specified load.

Configurations Measured

The following HA adapter configurations were measured to provide a clear picture of the performance/quality-of-service tradeoffs available in different configurations:

- Simple failover with no buffering
- Simple failover with a 15 second buffer (default buffer size)
- Lightweight queue trimming with trim interval of 100 events
- Lightweight queue trimming with trim interval of 1 event (trim on every event to minimize duplicates on failure)
- Precise recovery

Note that in each case the JMS topic used for output was configured with a quality of service that was appropriate for the CEP configuration being measured. CEP adapter configurations that have some potential for message loss on failure (simple failover) were measured using a non-persistent JMS configuration that could also experience message loss in the case of JMS server failure, while CEP configurations that guarantee no message loss (lightweight queue trimming and precise) were measured with JMS message persistence ensuring reliable delivery of the output even in the case of a JMS failure. This approach allows a performance comparison of different quality of service levels for the system as a whole, including the JMS output.

Metrics Collected

For each benchmark configuration, the following metrics were recorded:

- HA adapter configuration (determines HA Quality-of-Service)
- Number of partitions within a given server instance.
- Injection rate per server. This is the maximum injection rate that could be sustained in steady state over a 10 minute benchmark run.
- Server processing latency. This latency is measured only on the primary servers and only for events that result in output. An initial timestamp is taken in the adapter after reading the input data from the socket and prior to unmarshalling. A second timestamp is taken after the corresponding output event has been sent to JMS and the HA adapter has completed any processing associated with the event (e.g. trimming messages to the secondary). The difference between the timestamps is recorded as the latency for a particular event. Latencies are aggregated, and average and 99.99 percentile latencies are reported for each run.
- Output event rate

Hardware and Software Stack

The hardware environment consisted of a two machine Oracle CEP cluster with additional machines hosting the load generator and the external WebLogic JMS topic used for output. The two machines hosting the CEP cluster had similar, but not identical hardware specifications as described below:

Machine 1:

- 2 Intel Xeon X5670 processors at 2.93 GHz (6 cores each with hyperthreading – 24 total hardware threads)
- 72 GB Memory

Machine 2:

- 2 Intel Xeon X5660 processors at 2.80 GHz (6 cores each with hyperthreading – 24 total hardware threads)
- 36 GB Memory

The software stack for all CEP server instances was identical as follows:

- Oracle Enterprise Linux 5.4, kernel 2.6.18-164, x86_64
- Oracle JRockit JVM 1.6.0_20, 32 bit build
- Oracle Complex Event Processing 11.1.1.4

Benchmark Results

Table 1 shows the runtime performance results for various HA adapter configurations. The table shows the maximum steady state injection rate achievable in each of 5 different adapter configurations. Note that the granularity used when increasing the injection rate to determine the maximum was in increments of 1000 events per second per partition. As a result, any differences in maximum throughput that were smaller than this increment weren't detected.

Partitions per Server	Injection Rate/Server (events/sec)	Total Injection Rate	Output Rate (events/sec)	Average Latency (microsecs)	99.99% Latency (millisecs)
Simple Failover (no buffering)					
8	456,000	912,000	46,213	78.3	9.7
Simple Failover with Buffering (15 second buffer)					
8	376,000	752,000	38,110	95.4	28.1
Lightweight Queue Trimming with Trim Interval of 100					
8	184,000	368,000	18,597	453.1	20.5
Lightweight Queue Trimming with Trim Interval of 1 (trim on every event)					
8	184,000	368,000	18,504	560	21.1
Precise					
8	176,000	352,000	17,769	594.7	15.8

Table 1. Runtime Throughput and Latency

The following points are noteworthy in these results:

- There is a significant difference in both throughput and latency when comparing simple failover with no buffering to simple failover with a 15 second buffer. Even though there is no additional work done on the primary when buffering is configured, the secondary does incur some overhead in managing the buffer and also experiences additional garbage collection cost related to the buffer memory.

- There is measurable difference in performance of lightweight queue trimming when making a significant change to the trimming interval. Although the maximum throughput is the same for the two measured trimming intervals (within the injection granularity), the performance impact of more frequent trimming can be seen in the latency results. So there is a tradeoff in this case between performance, and the potential for duplicates on failure (which increases with an increased trim interval).
- The latencies are significantly higher and throughputs are lower for lightweight queue trimming and precise when compared to the simple failover configurations. A major factor contributing to the performance difference is the difference in how JMS output is handled. For lightweight queue trimming and precise recovery the primary makes a synchronous call to the JMS server for each output event, and the output processing isn't complete until JMS has persisted the message and sent a response acknowledging the reliable message receipt. In the simple failover configurations a lower quality of service was configured for JMS output as discussed earlier, and so output events are sent without waiting for a synchronous acknowledgement and the overhead of the roundtrip network latency to the JMS server and the JMS persistence costs are avoided.
- Looking at the overall relationship of performance vs. quality-of-service we see that performance improves with a lower quality-of-service as one would expect. The various adapter configurations provided by CEP allow users the flexibility to make the performance/QOS tradeoff that is appropriate for their environment and specific requirements.

Overall the various adapter configurations were able to demonstrate very high throughputs and low processing latencies in an HA configuration. These results validate the belief that the active-active approach has very good performance characteristics. In addition, the ability to scale both to multiple partitions (threads) within a server instance and multiple server instances suggests that the Oracle CEP HA approach will scale well as additional hardware resources are added.

CONCLUSIONS

This paper has described the general problem of providing high-availability for complex event processing applications and presented a comparison of the alternative approaches that are available. The active-active HA solution used by the Oracle CEP product was presented along with a discussion of the advantages of this approach and a benchmark study which validates the performance of this approach. The Oracle CEP approach to high-availability emphasizes performance, simplicity, and scalability.

Oracle CEP HA provides a number of alternatives that allow users to make the most appropriate performance vs. quality-of-service tradeoff for their particular environment, including a precise recovery option that guarantees no loss or duplication of output events. The paper covered a number of application design considerations that need to be taken into account by architects and developers of HA applications. In all cases, a simple best practice was presented for making the application highly available.

Oracle CEP HA addresses the problems that failure of an Oracle CEP instance pose for an end-to-end HA solution. In general, enterprises need to take a holistic view of high-availability and design applications to be highly available up front, rather than attempting to address high availability later in the application development lifecycle. Oracle CEP HA is designed to fit into a holistic approach to high-availability, and can help make applications both highly available and performant.

REFERENCES

- Marcus, E., and Stern, H., Blueprints For High-Availability, Second Edition, Wiley Publishing, 2003
- Oracle CEP product page on OTN
- Oracle CEP online product documentation
- Oracle Complex Event Processing and Distributed Caching, Oracle Whitepaper, December 2008
- Oracle Coherence product page on OTN

Oracle Complex Event Processing High Availability
November 2010

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com