IVrixDB Tests Overview

KEY THEME: Each test builds upon all the aspects and features of the previous tests. This means that the tests gradually become more **integrated** and ensures that the system can **function at scale**.

1. Indexing into IVrix Cluster
    a. Direct indexing into IVrix Node
    b. Load-Balanced Indexing into IVrix Cluster
    c. Node-Failure during Load-Balanced Indexing
        i. Forwarder Failure
        ii. Forwardee Failure
    d. Multiple batches for same index on one IVrix Node
    e. segregation of data into buckets with proper index rollover
2. Single Search on IVrix Cluster
    a. Hold Bucket, then Release once finished with Bucket
    b. Time-Ranged Search
        i. Hot
        ii. Warm
        iii. Cold
            1. attach/detach of COLD buckets
    c. Unbounded Search (Search Features in General)
    d. Keyword Search
    e. Statistics Search (Results page)
    f. Sort Search
    g. Sort + Statistics Search
    h. “SELECT” solr-function Search
    i. “HAVING” solr-function Search (filtering on field from STFE)
    j. Chained Search (w/ streaming and non-streaming commands)
3. Multiple Searches on IVrix Cluster
    a. Searches waiting on one another
4. Indexing + Single Search on IVrix Cluster (index rollover during search)
    a. All Cold being held during index rollover
    b. Oldest Warm being held during index rollover
5. Indexing + Multiple Searches on IVrix Cluster
6. Node Failure during Indexing + Searching on IVrix Cluster
    a. Node Failure
    b. Overseer-specific Failure
    c. Slave-specific Failure
    d. OPTIONAL TESTS --- operation failure/recovery/partial-failure/completion
7. Recovery of Node after Node failure during Indexing + Searching on IVrix Cluster

Tests Details

Setup for All Tests (unless explicitly stated for each point)

- Two IVrix Nodes, One Zookeeper Node
- 8.03 million mock logs, spanning over 11 years with 2000 events per day
    - As one file
    - As two files
        - Split into two files, in a round-robin batched separation method
- Config in "Constants" class:
    - DEFAULT_REPLICATION_FACTOR of 2 (for buckets)
    - MAXIMUM_ATTACHED_HOT_PLUS_WARM_BUCKETS_PER_NODE = 10
    - MAXIMUM_ATTACHED_COLD_BUCKETS_PER_NODE = 3
    - DEFAULT_MAX_EVENT_COUNT_IN_BUCKET = 401500
    - DEFAULT_EVENT_BATCH_SIZE = 30,000

Teardown

- Restart IVrix Nodes (not always though). This is done for two reasons that are further detailed in the "Disclaimers" section of "Disclaimers, Warnings, and Bugs" document:
    - Due to the bug where an IVrix Index cannot be fully deleted without a restart of the residing nodes, the IVrix Nodes must be restarted. Additionally, the directories of detached buckets still exist, and so manual removal of them is a must.
    - Due to no memory resource manager for search, memory just keeps growing and growing during run-time. That starts to affect the machine itself, and so periodic restarts are necessary.

1 – Indexing into IVrix Cluster

1A – Direct indexing into IVrix Node

- ***TEST 1:***
    - Procedures:
        - Index the one file with 8.03 million logs into one IVrix Node (w/o load-balancing)
    - Expected outcome:
        - One Bucket with 8.03 million logs

1B – Load-Balanced Indexing into IVrix Cluster

- ***TEST 1:***
    - Procedures:
        - Index the one file with 8.03 million logs into one IVrix Node (w/ load-balancing)
    - Expected outcome:
        - 20 buckets, 10 per node
        - 420k events per bucket (2_0k on last two buckets). Surpasses limit because of batch sizing

1C – Node-Failure during Load-Balanced Indexing

Forwarder Failure

- ***TEST 1:***
    - Procedures:
        - Index the one file with 8.03 million logs into IVrix Node 8983 (w/ load-balancing)
        - Kill IVrix Node 8983 unexpectedly
    - Expected outcome:
        - Forwarder will die. No more indexing from that request.

Forwardee Failure

- ***TEST 1 (During Indexing into Dead Node):***
    - Procedures:
        - Index the one file with 8.03 million logs into IVrix Node 8983 (w/ load-balancing)
        - Kill IVrix Node 7574 while forwarder sent batch to it
            - Add breakpoint at EventForwarder.tryToIndexBatch() on .request() line to assist in timing
- Expected outcome:
        - Forwarder will re-send batch to a live node
        - All Events will be indexed
        - Duplicate Events will likely exist (maximum of 30k duplicates)
- ***TEST 2 (During Indexing into Other Node):***
    - Procedures:
        - Index the one file with 8.03 million logs into IVrix Node 8983 (w/ load-balancing)
        - Kill IVrix Node 7574 while forwarder sent batch to 8983
            - Add breakpoint at EventForwarder.tryToIndexBatch() on .request() line to assist in timing
    - Expected outcome:
        - Forwarder will ignore dead node
        - All Events will be indexed

1D – Multiple batches for same index on one IVrix Node

- ***TEST 1:***
    - Procedures:
        - Index one half file into IVrix Node 8983 (w/ load-balancing)
        - Index the other half file into IVrix Node 7574 (w/ load-balancing)
    - Expected outcome:
        - 20 buckets, 10 per node
        - 420k events per bucket (2_0k on last two buckets). Surpasses limit because of batch sizing

1E – Segregation of Data into Buckets with Proper Index Rollover

- ***TEST 1:***
    - Setup:
        - MAXIMUM_ATTACHED_HOT_PLUS_WARM_BUCKETS_PER_NODE = 10

        - MAXIMUM_ATTACHED_COLD_BUCKETS_PER_NODE = 3
    - Procedures:
        - Index the one file with 8.03 million logs into IVrix Node 8983 (w/ load-balancing)
    - Expected outcome:
        - 20 buckets
        - (HOT, WARM, COLD, DETACHED)
        - Node 8983 -> (1, 4, 3, 2)
        - Node 7574 -> (1, 4, 3, 2)
- ***TEST 2:***
    - Setup:
        - MAXIMUM_ATTACHED_HOT_PLUS_WARM_BUCKETS_PER_NODE = 4
        - MAXIMUM_ATTACHED_COLD_BUCKETS_PER_NODE = 1
    - Procedures:
        - Index the one file with 8.03 million logs into IVrix Node 8983 (w/ load-balancing)
    - Expected outcome:
        - 20 buckets
        - (HOT, WARM, COLD, DETACHED)
        - Node 8983 -> (1, 1, 1, 7)
        - Node 7574 -> (1, 1, 1, 7)

2 – Single Search on IVrix Cluster

SETUP FOR ALL TESTS IN THIS SECTION:

- Have the index ready and fully indexed
- index the one file with 8.03 million logs into IVrix Node 8983 (w/ load-balancing) [**1B TEST 1, PROCEDURE**]

2A – Hold Bucket, then Release once finished with Bucket

- ***TEST:***
    - Procedures:
        - Run un-bounded search
    - Expected outcome:
        - For Every Hold of a group of buckets, there should be a release of them ***BEFORE*** the hold of the next group of buckets.

2B – Time-Ranged Search

- ***TEST 1:***
    - Procedures:
        - Run ranged search over HOT buckets
            - (_default20)
            - latest: *2010-12-29T02:46:40Z*, earliest: *2010-12-22T01:23:25Z*
        - Run ranged search over WARM buckets

                - (_default18, _default17, _default16, _default15)
                - latest: *2009-11-16T02:05Z*, earliest: *2008-10-12T00:00:05Z*
            - Run ranged search over COLD buckets
                - (_default4, _default3, _default2, _default1)
                - latest: *2002-03-13T01:23:20Z*, earliest: *2000-01-02T00:00:05Z*
    - Expected outcome:
        - HOT buckets – 15,000 events
        - WARM buckets – 801,500 events
        - COLD buckets – 1,603,000 events
            - Attaching/Detaching buckets appropriately
            - Maintaining attachment constraint limits

- ***TEST 2:***
    - Procedures:
        - Run ranged search over 1 Timeline bucket
            - latest: *2010-12-01T00:00Z*, earliest: *2010-11-01T00:00Z*
        - Run ranged search over 2 Timeline buckets
            - latest: *2010-12-01T00:00Z*, earliest: *2010-10-01T00:00Z*
        - Run ranged search over 4 Timeline buckets
            - latest: *2010-12-01T00:00Z*, earliest: *2010-08-01T00:00Z*
    - Expected outcome:
        - 1 Timeline bucket – 60,000 events
        - 2 Timeline bucket – 122,000 events
        - 4 Timeline bucket – 244,000 events

2C – Unbounded Search (Search Features in General)

- ***TEST:***
    - Procedures:
        - Run un-bounded search:
            - search(_default,zkHost="localhost:9983",ivrix=true,qt="/export",q="*:*",fl="_raw,_time,id",sort="_time desc",partitionKeys=_time)
    - Expected outcome:
        - Time range – over a span of 11 years (*2000-01-01T00:00Z* to *2011-01-01T00:00Z*)
        - Number of events – 8,030,000
        - Should attach/detach over COLD buckets when necessary
        - ***ALL SEARCH API COMPONENTS ARE INCREMENTALLY UPDATED OVER THE SEARCH PERIOD***
        - Timeline
            - Timeline will grow as events are digested in reverse time order (latest to earliest)
            - At the end
                - Timeline buckets in the scope of “MONTH”

            - each bucket has between 56k to 62k events
        - Events
            - All-inclusive bucket has only 1k events stored in it
            - Each subsequent bucket has only 1k events stored in it
            - Un-bounded pagination of events returns only 1k events
            - Bounded pagination of events returns **1k*N** events, depending on the number of buckets selected
        - Field Summaries
            - Number fields show stats (min, max, avg)
            - String and Number fields show top 10 values (count/percentage of each value)
        - Results
            - Empty

2D – Keyword Search

- ***TEST:***
    - Procedures:
        - Run un-bounded, keyword search (keyword "simpson"):
            - search(_default,zkHost="localhost:9983",ivrix=true,qt="/export",q="_raw:simpson",fl="_raw,_time,id",sort="_time desc",partitionKeys=_time)
    - Expected outcome:
        - Timeline, Events, and Field Summaries just as **2C**, **EXCEPT**:
            - Timeline
                - Number of events – 1,604,004
                - Each bucket has between 10k to 13k events
        - All events must have keyword "simpson" in them

2E – Statistics Search (Results page)

- ***TEST:***
    - Procedures:
        - Run un-bounded, statistics search (group by over "age", sum over "random"):
            - rollup(search(_default,zkHost="localhost:9983",ivrix=true,qt="/export",q="*:*",fl="_raw,_time,id",sort="_time desc",partitionKeys=_time),over=age,sum(random))
    - Expected outcome:
        - Timeline, Events, and Field Summaries just as **2C**
        - Results
            - ***INCREMENTALLY UPDATED OVER THE SEARCH PERIOD***
            - Groups over age of 10, 20, 30, 40, 50, 60, and 70
            - Each group has an ever-increasing value of "sum(random)"

2F – Sort Search

- ***TEST:***
    - Procedures:
        - Run un-bounded, sort search (top 10k, random desc):

- top(n=10000,search(_default,zkHost="localhost:9983",ivrix=true,qt="/export",q="*:*",fl="_raw,_time,id",sort="_time desc",partitionKeys=_time),sort="random desc")

- Expected outcome:
  - Timeline, Events, and Field Summaries just as **2C**, **EXCEPT**:
    - Timeline
      - Number of events – 10,000. This number does **NOT** change as the search goes on
      - Number of events per timeline bucket – changes frequently, and gets smaller and smaller as the search goes on
    - Events
      - Same "page" has a strong likelihood of changing after each refresh

2G – Sort + Statistics Search

- ***TEST:***
  - Procedures:
    - Run un-bounded, sort + statistics search (top 10k, sorted by random DESC, THEN sum over random):
      - rollup(top(n=10000,search(_default,zkHost="localhost:9983",ivrix=true,qt="/export",q="*:*",fl="_raw,_time,id",sort="_time desc",partitionKeys=_time),sort="random desc"),over=age,sum(random))
  - Expected outcome:
    - Timeline, Events, and Field Summaries just as **2F**
    - Results just as **2E**, except that "sum(random)" is not ever-increasing by large changes, rather it is ever-changing

2H – "SELECT" solr-function Search

- ***TEST:***
  - Procedures:
    - Run unbounded, "SELECT" solr-function Search (replace all instances where age=70 with value 1000):
      - select(search(_default,zkHost="localhost:9983",ivrix=true,qt="/export",q="*:*",fl="_raw,_time,id",sort="_time desc",partitionKeys=_time),age,_time,_raw,weight,random,replace(age,70,withValue=1000))
  - Expected outcome:
    - Timeline, Events, and Field Summaries just as **2C**, **EXCEPT**:
      - Events where age=70, the value is replaced with 1000

2I – "HAVING" solr-function Search (filtering on field from STFE)

- ***TEST:***
  - Procedures:

        - Run unbounded, “HAVING” solr-function Search (filtering on STFE field “age”, where age = 10)
            - having(search(_default,zkHost="localhost:9983",ivrix=true,qt="/export",q="*:*",fl="_raw,_time,id",sort="_time desc",partitionKeys=_time),eq(age,10))
    - Expected outcome:
        - Timeline, Events, and Field Summaries just as **2C**, **EXCEPT**:
            - Timeline
                - Number of events – 1,146,982
                - Each bucket has less than 10k (between 8k to 10k) events
            - Events
                - There are **ONLY** events with age = 10

2J – Chained Search (w/ streaming and non-streaming commands)

- ***TEST:***
    - Procedures:
        - Run unbounded, keyword + SELECT + sort + HAVING + statistics search
            - rollup(having(top(n=10000,select(search(_default,zkHost="localhost:9983",ivrix=true,qt="/export",q="_raw:simpson",fl="_raw,_time,id",sort="_time desc",partitionKeys=_time),age,_time,_raw,weight,random,replace(age,70,withValue=1000)),sort="random desc"),eq(age,10)),over=age,sum(random))
    - Expected outcome:
        - Timeline
            - Number of events – 1,445
            - Each bucket has less than 20 events
        - Events
            - Have only fields age, _time, _raw, weight, and random
            - There are **ONLY** events with age = 10
        - Results
            - One group (age of 10)
            - “sum(random)” is not ever-increasing by large changes, rather it is ever-changing

3 – Multiple Searches on IVrix Cluster

SETUP FOR ALL TESTS IN THIS SECTION:

- Have the index ready and fully indexed

- index the one file with 8.03 million logs into IVrix Node 8983 (w/ load-balancing) [**1B TEST 1, PROCEDURE**]

3A – Searches waiting on one another

- ***TEST:***
    - Procedures:
        - Run 4 ranged searches simultaneously spanning over the COLD buckets (half running on Node 8983, half running on Node 7574)
            - 7574 #1 -- latest: *2006-05-22T02:46:40Z*, earliest: *2004-05-23T02:46:40Z*
            - 8983 #1 -- latest: *2004-05-22T02:46:40Z*, earliest: *2002-05-02T01:23:25Z*
            - 8983 #2 -- latest: *2003-04-18T00:41:40Z*, earliest: *2001-03-27T02:05:05Z*
            - 7574 #2 -- latest: *2002-03-13T01:23:20Z*, earliest: *2000-01-02T00:00:05Z*
    - Expected outcome:
        - Some Searches will need to wait to attach, since not all buckets can be attached at once
        - 7574 #1 -- 1458001 events
        - 8983 #1 -- 1503000 events
        - 8983 #2 -- 1503000 events
        - 7574 #2 -- 1603000 events

4 – Indexing + Single Search on IVrix Cluster (index rollover during search)

SETUP FOR ALL TESTS IN THIS SECTION:

- MAXIMUM_ATTACHED_HOT_PLUS_WARM_BUCKETS_PER_NODE = 4
- MAXIMUM_ATTACHED_COLD_BUCKETS_PER_NODE = 1

4A – All Cold Being Held During Index Rollover

- ***TEST:***
    - Procedures:
        - Add breakpoint at IVrixOverseer.executeCreateBucketOperation() before the state lock command
            - With condition: state.getIndex(indexName).getBucket("_default7") == null && state.getIndex(indexName).getBucket("_default6") != null
        -

        - index the one file with 8.03 million logs into IVrix Node 8983 (w/ load-balancing) [**1B TEST 1, PROCEDURE**]
        - Once breakpoint is hit, execute search on (_default1, _default2)
            - latest: 2000-07-20T00:00:05Z, earliest: 2000-01-02T00:00:05Z
        - Release breakpoint
    - Expected outcome:
        - Before Search finishes:
            - _default3 is rolled to cold, and detached
            - _default7 is created

4B – Oldest Warm Being Held During Index Rollover

- ***TEST:***
    - Procedures:
        - Add breakpoint at IVrixOverseer.executeCreateBucketOperation() before the state lock command
            - With condition: state.getIndex(indexName).getBucket("_default5") == null && state.getIndex(indexName).getBucket("_default4") != null
        -
        - index the one file with 8.03 million logs into IVrix Node 8983 (w/ load-balancing) [**1B TEST 1, PROCEDURE**]
        - Once breakpoint is hit, execute search on (_default1, _default2)
            - latest: *2000-07-20T00:00:05Z,* earliest*: 2000-01-02T00:00:05Z*
        - Release breakpoint
    - Expected outcome:
        - Waits for search on _default1 to finish
        - In the meantime, no new searches on _default1 are created, even if requested
        - Once search on _default1 finishes
            - _default 1 rolls to cold
            - _default5 is created

5 – Indexing + Multiple Searches on IVrix Cluster

- ***TEST:***
    - Procedures:
        - index the one file with 8.03 million logs into IVrix Node 8983 (w/ load-balancing) [**1B TEST 1, PROCEDURE**]
        - Wait until bucket _default16 is made
        - Run 4 ranged searches simultaneously spanning over the COLD buckets (half running on Node 8983, half running on Node 7574)
            - 7574 #1 -- latest: *2006-05-22T02:46:40Z*, earliest: *2004-05-23T02:46:40Z*

        - 8983 #1 -- latest: *2004-05-22T02:46:40Z*, earliest: *2002-05-02T01:23:25Z*
        - 8983 #2 -- latest: *2003-04-18T00:41:40Z*, earliest: *2001-03-27T02:05:05Z*
        - 7574 #2 -- latest: *2002-03-13T01:23:20Z*, earliest: *2000-01-02T00:00:05Z*
    - Expected outcome:
        - Some Searches will need to wait to attach, since not all buckets can be attached at once
        - When Indexing needs to roll oldest warm bucket, but it is being held
            - Waits for search on bucket to finish
            - In the meantime, no new searches on bucket are created, even if requested
        - When Indexing needs to make space for cold bucket, but all cold is being held
            - Detaches that cold bucket
        -
        - 7574 #1 -- 1458001 events
        - 8983 #1 -- 1503000 events
        - 8983 #2 -- 1503000 events
        - 7574 #2 -- 1603000 events
        - Indexing will finish with all 20 buckets filled, and all 8.03 million logs

6 – Node Failure during Indexing + Searching on IVrix Cluster

- ***TESTS 1 & 2:***
    - Procedures:
        - index the one file with 8.03 million logs into either Node 8983 or Node 7574 (w/ load-balancing)
        - Wait until bucket _default17 is made
        - Run 4 ranged searches simultaneously spanning over the COLD buckets (half running on Node 8983, half running on Node 7574)
            - 7574 #1 -- latest: *2006-05-22T02:46:40Z*, earliest: *2004-05-23T02:46:40Z*
            - 8983 #1 -- latest: *2004-05-22T02:46:40Z*, earliest: *2002-05-02T01:23:25Z*
            - 8983 #2 -- latest: *2003-04-18T00:41:40Z*, earliest: *2001-03-27T02:05:05Z*
            - 7574 #2 -- latest: *2002-03-13T01:23:20Z*, earliest: *2000-01-02T00:00:05Z*
            - *8983 #3 – UNBOUNDED*
            - *7574 #3 -- UNBOUNDED*
        - Drop either Node 8983 or Node 7574 unexpectedly

        - Re-run searches from dropped Node
    - Expected outcome
        - Operation recovery
            - Dropped Node is Overseer
                - Failure of all operations
                - Re-election of New Overseer
                - New Overseer executes recovery from failed physical operations
                - Running Searches will re-hold buckets***(**not implemented**)
                - re-sending of operations to new overseer
            - Dropped Node is Slave
                - Completion or partial completion of physical operations utilizing slave node
                - Overseer Recovery from partially completed physical operations
        -
        - Failure of all Search running on dropped Node
            - Overseer removes Bucket Holders originating from that Node
        - Failure of all Indexing on dropped Node
            - Replication will be paused, and Indexer replicas will no longer be indexed into
        - Continuation of Indexing and Searching on all live nodes
            - Indexing will continue, even though replication will not occur on the dead node
                - Failed Batches will be redirected to the live node
                - All data will be indexed
            - Searching will continue or fail, depending on whether a replica that it was searching on died, or whether a bucket cannot be physically held
        - New Searches on live nodes
            - Some will work, some will not because necessary buckets cannot be retrieved

(OPTIONAL) -- MORE DETAILED TESTS REGARDING OPERATION FAILURE & RECOVERY:

Operation Recovery from Overseer Death

- ***CREATE BUCKET***
    - ***TEST 1, 2, 3, & 4***
        - Procedure
            - Add breakpoint at IVrixBucket.createSelf(), on either line
                - createIsDetachedPropertyNode()
                - createBucketTypePropertyNode()
                - createBucket()
                - updateSolrMetadata()

                - hit breakpoint, then fail overseer
            - Expected Outcome
                - When new overseer boots, it will deem the bucket as dead/unusable, and will delete it
- ***ROLL TO COLD***
    - ***TEST 1 & 2***
        - Procedure
            - Add breakpoint at IVrixBucket.rollToCold(), on either line
                - prepareWarmBucketForColdRollover()
                - updateSolrMetadata()
            - hit breakpoint, then fail overseer
        - Expected Outcome
            - When new overseer boots, it will fully roll the bucket to COLD (i.e., remove all unnecessary replicas and update the solrMetadata field in the bucket)
- ***ATTACH***
    - ***TEST 1***
        - Procedure
            - Add breakpoint at IVrixBucket.physicallyAttachOnly()
            - hit breakpoint, then fail overseer
        - Expected Outcome
            - When new overseer boots, it will fully attach the bucket
- ***DETACH***
    - ***TEST 1***
        - Procedure
            - Add breakpoint at IVrixBucket.physicallyDetachOnly()
            - hit breakpoint, then fail overseer
        - Expected Outcome
            - When new overseer boots, it will fully detach the bucket

Operation Recovery from Slave Death

- ***CREATE BUCKET***
    - ***TEST 1***
        - Procedure
            - Add breakpoint at createAddReplicaRequest(...indexerNodeName...)
            - fail node, then release breakpoint
        - Expected Outcome
            - Bucket will fail to create, and will be a hanging/dead bucket
            - When overseer receives notification of dead node, it will find the hanging/dead bucket, and delete it
    - ***TEST 2***
        - Procedure
            - Add breakpoint at createAddReplicaRequest(...replicationNodeName...)
            - fail node, then release breakpoint

        - Expected Outcome
            - Replication replica will fail to be created
            - Bucket will continue to be created as normal, since failure to add replication replica does not impact indexing
- ***ROLL TO COLD***
    - Cannot fail…
- ***ATTACH***
    - ***TEST 1***
        - Procedure
            - Add breakpoint at IVrixBucket.physicallyAttachOnly()
            - fail node, then release breakpoint
        - Expected Outcome
            - Failure to attach, but will not throw error
            - TRY TO HOLD BUCKET command will return "false", since it could not get a physical hold of the bucket
- ***DETACH***
    - Cannot fail…

7 – Recovery of Node after Node failure during Indexing + Searching on IVrix Cluster

- ***TEST 1:***
    - Procedures:
        - {procedures from **section 6**}
        - Re-boot dropped Node
        - Run New Searches from that Node
    - Expected Outcome
        - Buckets
            - Replication Replicas will sync up with other replicas
            - Previous Hot Buckets should not exist
            - Indexer Replicas will have their state recovered (time bounds) ***(**not implemented**)
        - State remains consistent, both on IVrixDB and Solr
        - currently executing Indexing and Searches will start to utilize recovered Node
        - New Searching commands can be executed that will utilize recovered Node
        - New Indexing commands can be executed that will utilize recovered Node