Cross Data Center Replication in Apache Solr

Anvi Jain, Software Engineer II Progress®
Amrit Sarkar, Search Engineer Lucidworks

Progress

Who are we?

Based in Bedford, MA. Offices all around the world

Progress tools and platforms enable the world’s leading businesses to deliver new technologies that are adaptive, connected and cognitive.

Solutions for enterprise integration, data interoperability and application development, including SaaS

Based in San Francisco. Offices in Cambridge, Bangalore, Bangkok, New York City, Raleigh, Munich

Over 300 customers across the Fortune 1000

Fusion, a Solr-powered platform for search-driven apps

Consulting and support for organizations using Solr

Lucidworks

Agenda

- Introduction to Apache Solr
- Motivation
  - Disaster Recovery Strategies available in Apache Solr
- Cross Data Center Replication
  - Uni-directional approach
  - Components
  - Bi-directional approach
  - Demonstration
  - Caveats & Limitations
  - Forthcoming improvements

Apache Solr

Solr

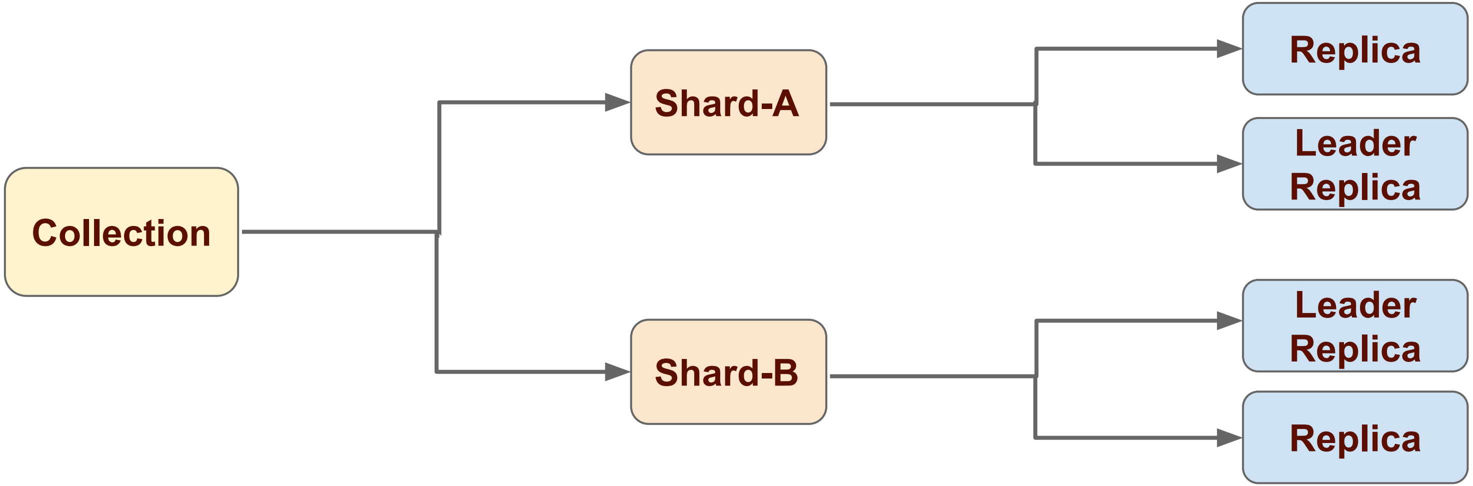
The standard for enterprise search.

- Apache Lucene
- Full text search
- Facets/Guided Nav galore!
- Lots of data types
- Spelling, autocomplete, highlighting
- More Like This
- Deduplication
- Grouping and Joins
- Stats, expressions, transformations
- Learning to Rank
- Extensible
- Massive Scale/Fault tolerance

Solr - SolrCloud

Progress Lucidworks

● Zookeeper:	Centralized service for maintaining configuration and cluster state info
● Collection:	A complete logical index in a SolrCloud
● Shard:	A logical piece (or slice) of a collection
● Replica:	One copy of a shard



Disaster Recovery

Major fire at a Samsung data center in Gwacheon, South Korea, 2014

- No data center is completely disaster proof.
- USA reported 1.3M and Germany 192K fire incidents on 2015 [CTIF World Fire Stats 2018]
- Disaster is not necessarily natural event, could be result of human errors, malicious acts (internal or external) or corrupt data.

Disaster Recovery

- Smaller companies less likely to have disaster recovery strategy implemented.
- Economy of scale is a challenge.
- Maintenance of “hot standby” is costly.
- Adopted strategy should be reliable and revisited regularly.

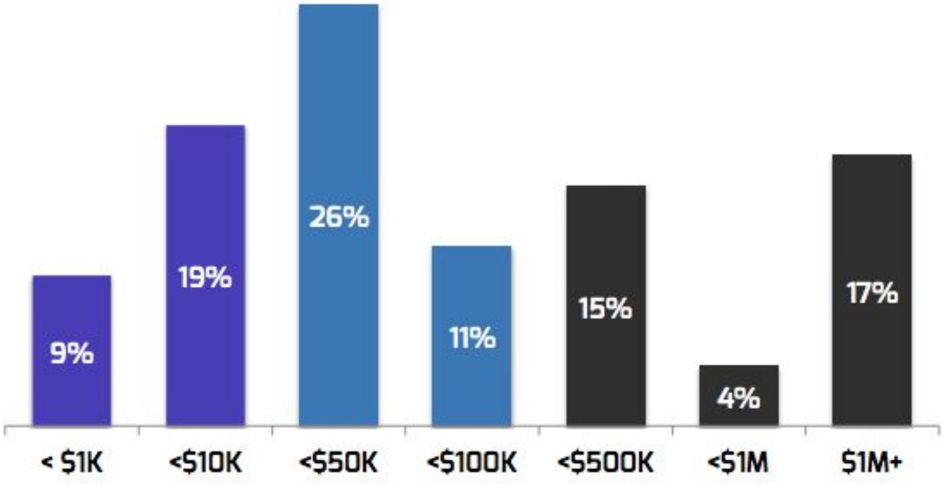


Figure 25: Cost of one day of downtime

2016 Disaster Recovery Survey

Solr - Disaster Recovery Strategies Available

Option	Risk
Index to two instances in different data centers	❖ Often require additional custom development (may need messaging service like Apache Kafka) ❖ No guarantee that the instances are identical
Disk Mirroring	❖ What if entire index file is not copied? ❖ What state is the disk at the time of abrupt event? ❖ In either case, entire index gets corrupted
Regular Backups	❖ Works if you have low volume index updates with controlled schedule ❖ Managing backups, storing offsite and retrieving quickly when needed is a challenge ❖ If backup is incomplete, index gets corrupted
Backup API	❖ Available in Solr at collection level ❖ Takes backup of entire data of collection along with configurations ❖ Prone to failure at live indexing and abrupt event

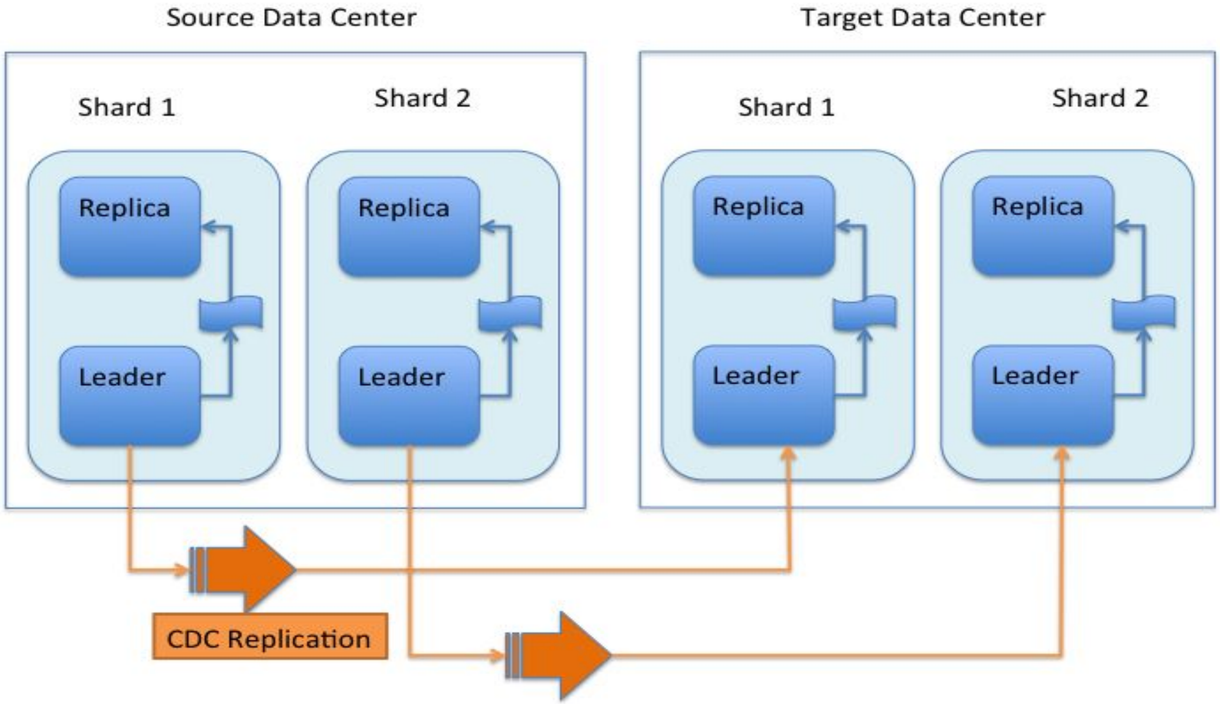
Cross Data Center Replication

in Apache Solr

Cross Data Center Replication

- Introduced in version 6.0.
- Forwarding updates/actions to secondary cluster(s) with ability to monitor and track replication and to see the clusters are healthy and running.
- Supports both unidirectional (active-passive) and bidirectional (active-active) approach.
- At Collection level, data is replicated, configurations are not.
- APIs available: */cdcr/action=START*, */cdcr/action=STOP*, */cdcr/action=STATUS*

CDCR Unidirectional Approach



Unidirectional approach in CDCR
(Active - Passive Clusters)

CDCR Unidirectional Configuration

Source collection configuration

```
<requestHandler name="/cdcr" class="solr.CdcrRequestHandler">
 <lst name="replica">
   <str name="zkHost">target_zk_host:2181</str>
   <!--
   If you have chrooted your Solr information at the target you must include the chroot, for
example:
   <str name="zkHost">10.240.18.211:2181,10.240.18.212:2181/solr</str>
   -->
   <str name="source">source_col</str>
   <str name="target">target_col</str>
 </lst>

 <lst name="replicator">
   <str name="threadPoolSize">8</str>
   <str name="schedule">1000</str>
   <str name="batchSize">128</str>
 </lst>

 <lst name="updateLogSynchronizer">
   <str name="schedule">1000</str>
 </lst>

</requestHandler>

<!-- Modify the <updateLog> section of your existing <updateHandler>
    in your config as below -->
<updateHandler class="solr.DirectUpdateHandler2">
 <updateLog class="solr.CdcrUpdateLog">
   <str name="dir">${solr.ulog.dir:}</str>
   <!--Any parameters from the original <updateLog> section -->
 </updateLog>

 <!-- Other configuration options such as autoCommit should still be present -->
</updateHandler>
```

Target collection configuration

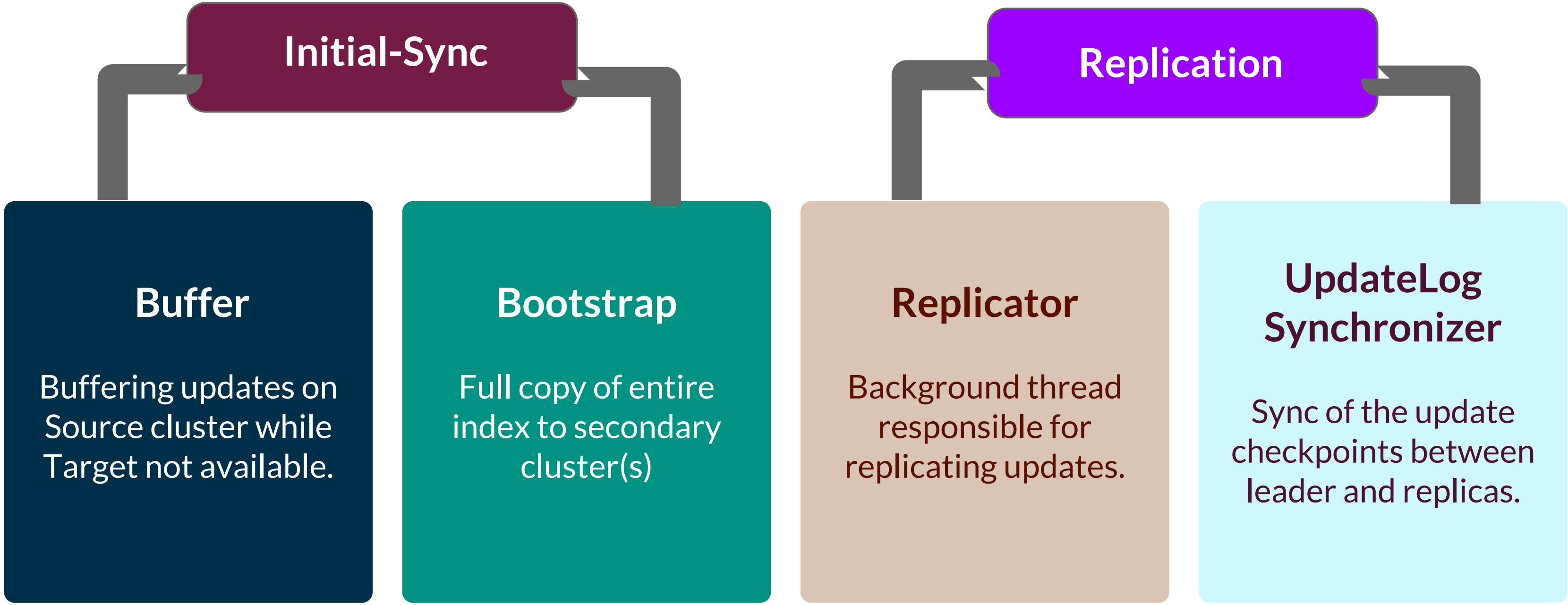
```
<requestHandler name="/cdcr" class="solr.CdcrRequestHandler">
  <!-- recommended for Target clusters -->
  <lst name="buffer">
    <str name="defaultState">disabled</str>
  </lst>
</requestHandler>

<requestHandler name="/update" class="solr.UpdateRequestHandler">
  <lst name="defaults">
    <str name="update.chain">cdcr-processor-chain</str>
  </lst>
</requestHandler>

<updateRequestProcessorChain name="cdcr-processor-chain">
  <processor class="solr.CdcrUpdateProcessorFactory"/>
  <processor class="solr.RunUpdateProcessorFactory"/>
</updateRequestProcessorChain>

<!-- Modify the <updateLog> section of your existing <updateHandler> in your
    config as below -->
<updateHandler class="solr.DirectUpdateHandler2">
  <updateLog class="solr.CdcrUpdateLog">
    <str name="dir">${solr.ulog.dir:}</str>
    <!--Any parameters from the original <updateLog> section -->
  </updateLog>

  <!-- Other configuration options such as autoCommit should still be present -->

</updateHandler>
```

CDCR Underlying Concept

- Updates are forwarded from leader nodes of source data center to leader nodes of target data center at regular intervals.
- Each update operation contains unique ‘version’, time-based import clock.

  e.g. _version_ : 1601710449509793792
- The unique ‘version’ is maintained as checkpoint, as ephemeral node on source and persistent node on target across all solr nodes.
- Does not require any additional network traffic.
- The update workflow of SolrCloud ensures every update is applied to the leader and its replicas, both on source and target.

CDCR Core Components



Initial-Sync

Replication

Buffer

Buffering updates on Source cluster while Target not available.

Bootstrap

Full copy of entire index to secondary cluster(s)

Replicator

Background thread responsible for replicating updates.

UpdateLog Synchronizer

Sync of the update checkpoints between leader and replicas.

CDCR Initial-Synchronization

- Updates are sent to the Source when Target is not available
- Target is either not created or no live nodes serving.

Initial-Sync: Synchronize data to Target when CDCR is started or resumed.

- Buffer
- Bootstrap

CDCR Buffer

```
<requestHandler name="/cdcr"
class="solr.CdcrRequestHandler">

................
  <lst name="buffer">
   <str

         name="defaultState">

         enabled

    </str>

..............
  </lst>
</requestHandler>
```

- References to updates will be kept in memory if ‘defaultState’ for ‘buffer’ is ‘*enabled*’ unless they are forwarded.
- Best to disable buffering during normal operation.
- API available:

solr/<collection>/cdcr?action=DISABLEBUFFER
solr/<collection>/cdcr?action=ENABLEBUFFER

CDCR Bootstrap

- If target checkpoint is negative i.e. no updates are received or too far behind source, bootstrap is triggered.
- Entire data directory of leader core of each shard of source is copied to leader core of each shard of target.
- After copy finishes, REQUESTRECOVERY API i.e. copy index from leader, is issued to non-leader cores of each shard of target.
- Index copy is single-threaded process, may take time for large indexes.
- Implicit API available, fire on target:

  solr/<collection>/cdcr?action=BOOTSTRAP&masterUrl=<leader in source dc URL>

CDCR State Managers

cdcr-state:

```
  ["process","started",
      "buffer","disabled"]
```

state.json:

```
{"citibike":{
  "pullReplicas":"0",
  "replicationFactor":"1",
  "shards":{"shard1":{
    "state":"active",
    "replicas":{"core_node2":{
     "core":"citibike_shard1_replica_n1",
"base_url":"http://cluster-1:8983/solr",
"node_name":"cluster-1:8983_solr",
"leader":"true"}}}},
  "router":{"name":"compositeId"}}
```

State managers are responsible to keep continuous watch on zookeeper nodes and trigger an action if changes are observed.

- **ProcessStateManager** keeps watcher on top of */collections/<name>/cdcr/state*
- **LeaderStateManager** keeps a watcher on top of */collections/<name>/state.json*

Informs replicator at state change.

CDCR Replicator

```
<requestHandler name="/cdcr"
class="solr.CdcrRequestHandler">
……………………..
  <lst name="replicator">
    <str
name="threadPoolSize">8</str>
    <str
name="schedule">1000</str>
    <str
name="batchSize">128</str>
  </lst>
        ……………………..
</requestHandler>
```

- Updates forwarded in batches at intervals specified in solrconfig.xml
- Single thread for each target is created for replication, unless specified explicitly in solrconfig.xml
- After each successful forward operation, checkpoints are updated both at target and at source.
- If leader node goes down, the newly elected leader resumes replication.
- If target is not available or down, source will retry indefinitely with same batch until connection is restored.

CDCR UpdateLogSynchronizer

- **Sharing checkpoint** at source across all replicas.
- Each non-leader will request **latest checkpoint** from its leader at regular intervals configured in solrconfig.xml.
- If current leader node goes down; the newly elected leader will replay updates in same order from its last ‘checkpoint’ leading to no loss of data.

```
<requestHandler name="/cdcr"
class="solr.CdcrRequestHandler">

…………….
  <lst
name="updateLogSynchronizer">
    <str name="schedule">

         60000

   </str>

…………..
  </lst>
</requestHandler>
```

CdcrUpdateProcessorFactory

- Must be defined at target data center if CDCR is enabled.
- **Skips** adding local version to avoid overriding incoming request containing ‘version’ from source.

```
<requestHandler name="/update"
class="solr.UpdateRequestHandler">
  <lst name="defaults">
    <str name="update.chain">
cdcr-processor-chain

   </str>
  </lst>
</requestHandler>

<updateRequestProcessorChain
name="cdcr-processor-chain">
  <processor
class="solr.CdcrUpdateProcessorFactory"/>
  <processor
class="solr.RunUpdateProcessorFactory"/>
</updateRequestProcessorChain>
```

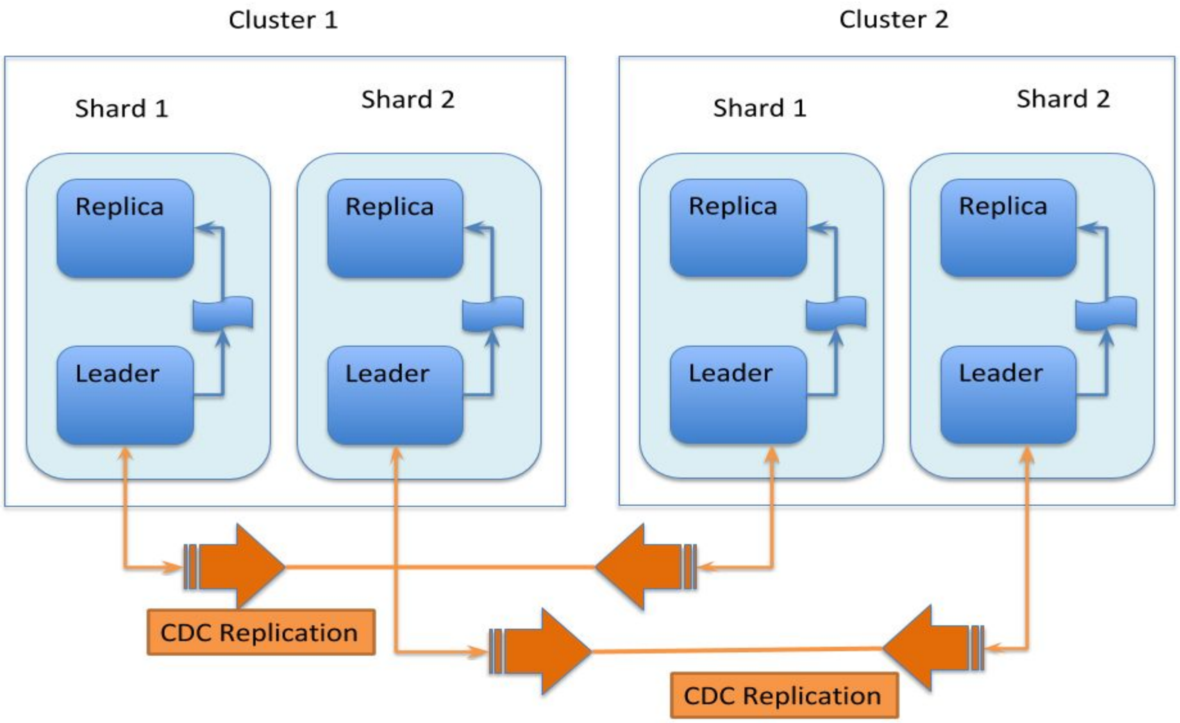
CDCR Monitoring API

API	Usage
• QUEUES: solr/<core>/cdcr?action=QUEUES	- Output composed of “queues” for each target data center *queueSize*: current size of the queue *lastTimestamp*: timestamp of last update operation - Also contains information about the update logs: *tlogTotalSize*: size of transaction logs on disk (in bytes) *tlogTotalCount*: number of transaction log files *updateLogSynchronozer*: status, either ‘running’ or ‘stopped’
• OPERATIONS: solr/<core>/cdcr?action=OPS	- *Average number of operations per second* and broken down by adds / deletes
• ERRORS: solr/<core>/cdcr?action=ERRORS	- The *number of consecutive errors* encountered - The *number of bad requests or internal errors* - *List of the last errors encountered* ordered by timestamp

CDCR Bidirectional Approach

- In unidirectional scenario
  - indexing can be done at one data center termed as source, will act as primary cluster
  - target data center, acting as secondary cluster, will be available for querying only.
- In version 7.2, bidirectional support was introduced
- Updates can be sent to either cluster and data will be synchronized

CDCR Bidirectional Approach



Bidirectional approach in CDCR (Active - Active Clusters)

CDCR Bidirectional Configuration

Progress Lucidworks

Cluster 1 collection configuration

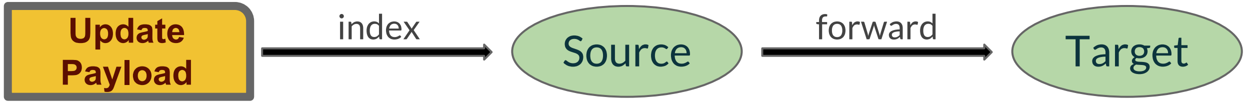
```
<requestHandler name="/update" class="solr.UpdateRequestHandler">
 <lst name="defaults">
  <str name="update.chain">cdcr-processor-chain</str>
 </lst>
</requestHandler>

<updateRequestProcessorChain name="cdcr-processor-chain">
 <processor class="solr.CdcrUpdateProcessorFactory"/>
 <processor class="solr.RunUpdateProcessorFactory"/>
</updateRequestProcessorChain>

<requestHandler name="/cdcr" class="solr.CdcrRequestHandler">
 <lst name="replica">
  <str name="zkHost">cluster2_zk_host:2181</str>
  <!--
  If you have chrooted your Solr information at the target you must include the chroot, for example:
  <str name="zkHost">10.240.19.241:2181,10.240.19.242:2181/solr</str>
  -->
  <str name="source">cluster1</str>
  <str name="target">cluster2</str>
 </lst>

 <lst name="replicator">
  <str name="threadPoolSize">8</str>
  <str name="schedule">1000</str>
  <str name="batchSize">128</str>
 </lst>

 <lst name="updateLogSynchronizer">
  <str name="schedule">1000</str>

</requestHandler>

<!-- Modify the <updateLog> section of your existing <updateHandler>
   in your config as below -->
<updateHandler class="solr.DirectUpdateHandler2">
 <updateLog class="solr.CdcrUpdateLog">
  <str name="dir">${solr.ulog.dir:}</str>
  <!--Any parameters from the original <updateLog> section -->
 </updateLog>
</updateHandler>
```

Cluster 2 collection configuration

```
<requestHandler name="/update" class="solr.UpdateRequestHandler">
 <lst name="defaults">
  <str name="update.chain">cdcr-processor-chain</str>
 </lst>
</requestHandler>

<updateRequestProcessorChain name="cdcr-processor-chain">
 <processor class="solr.CdcrUpdateProcessorFactory"/>
 <processor class="solr.RunUpdateProcessorFactory"/>
</updateRequestProcessorChain>

<requestHandler name="/cdcr" class="solr.CdcrRequestHandler">
 <lst name="replica">
  <str name="zkHost">cluster1_zk_host:2181</str>
  <!--
  If you have chrooted your Solr information at the target you must include the chroot, for example:
  <str name="zkHost">10.240.19.241:2181,10.240.19.242:2181/solr</str>
  -->
  <str name="source">cluster2</str>
  <str name="target">cluster1</str>
 </lst>

 <lst name="replicator">
  <str name="threadPoolSize">8</str>
  <str name="schedule">1000</str>
  <str name="batchSize">128</str>
 </lst>

 <lst name="updateLogSynchronizer">
  <str name="schedule">1000</str>

</requestHandler>

<!-- Modify the <updateLog> section of your existing <updateHandler>
   in your config as below -->
<updateHandler class="solr.DirectUpdateHandler2">
 <updateLog class="solr.CdcrUpdateLog">
  <str name="dir">${solr.ulog.dir:}</str>
  <!--Any parameters from the original <updateLog> section -->
 </updateLog>
</updateHandler>
```

CDCR Update Strategy

Unidirectional Approach



Update Payload

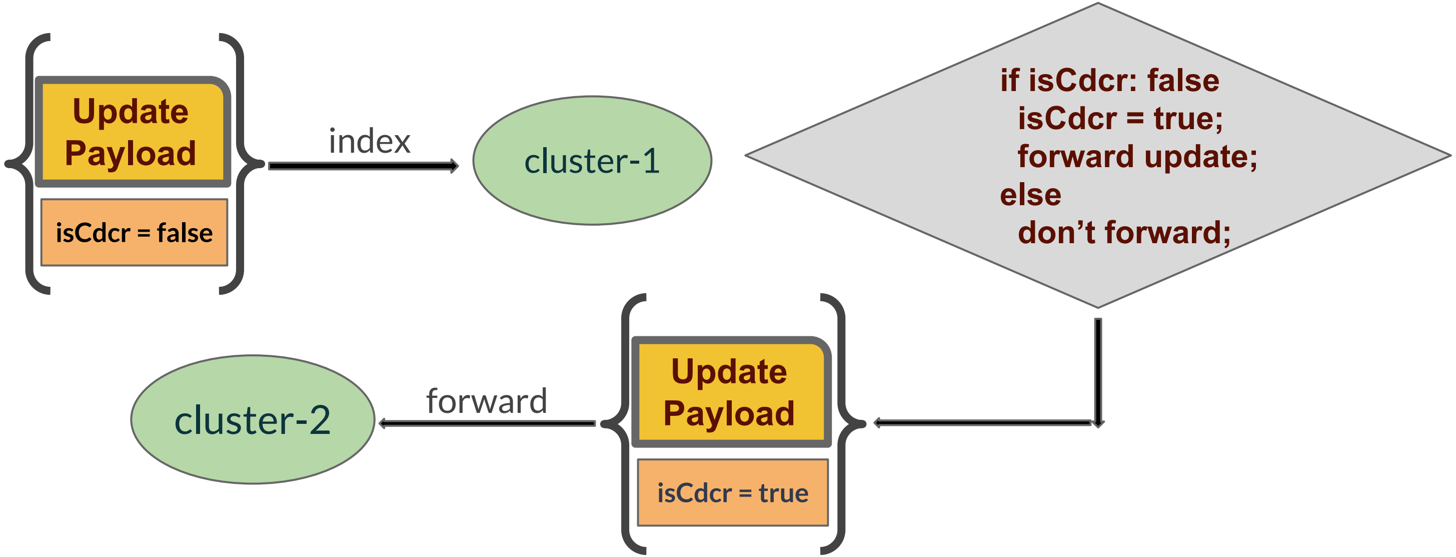
index

Source

forward

Target

Bidirectional Approach



Update Payload

isCdcr = false

index

cluster-1

```
if isCdcr: false
  isCdcr = true;
  forward update;
else
  don't forward;
```

Update Payload

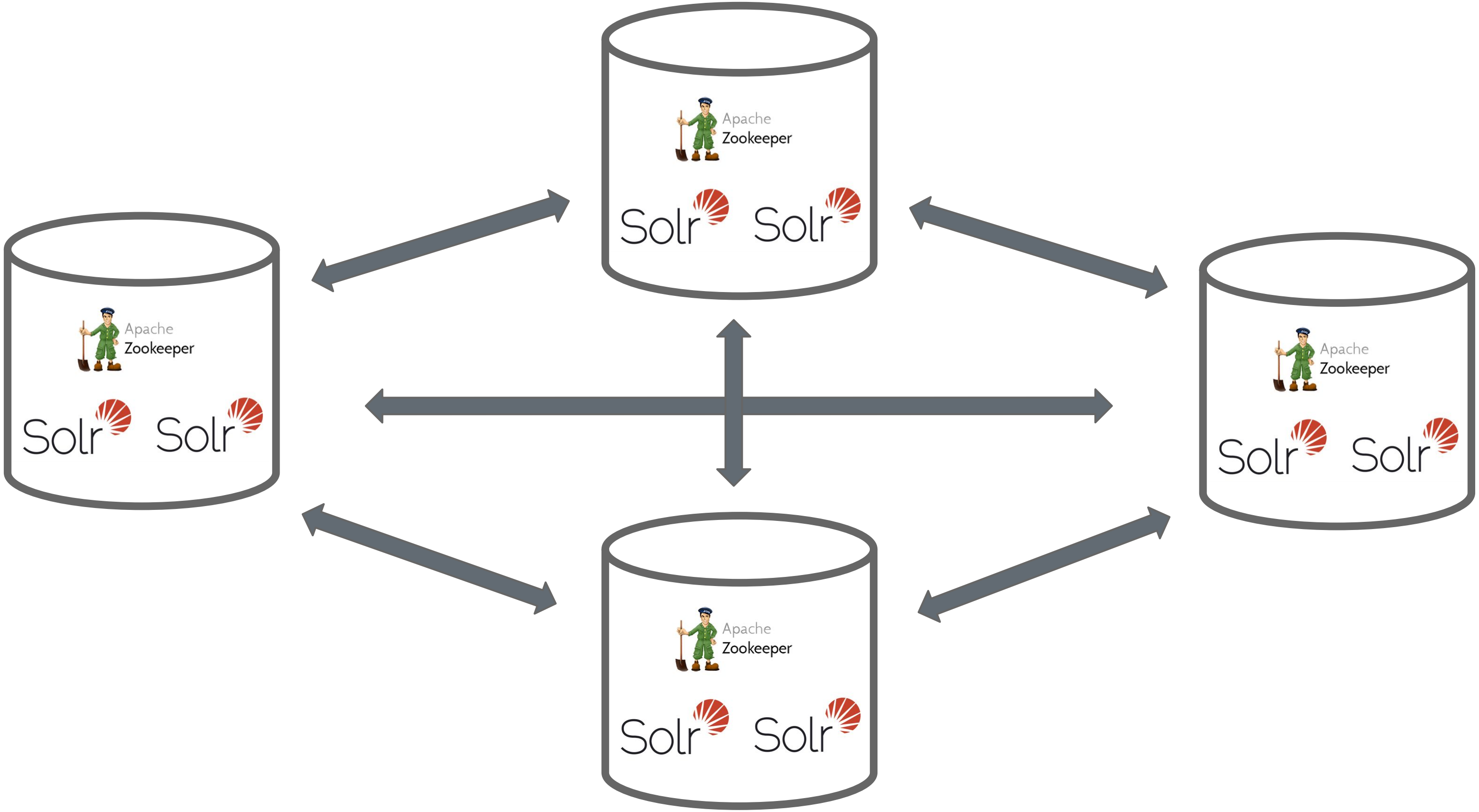
isCdcr = true

forward

cluster-2

Decentralised Multi-Cluster Setup

Progress® Lucidworks



CDCR Bidirectional Approach

- Decentralised, no source nor target
- No single point of failure anymore
- Indexing can be redirected to failover clusters swiftly without much human effort.
- Overall traffic can be distributed to multiple data centers based on their geographical location to avoid network delay, not applicable for real-time.

Demonstration

Caveats & Limitations

- Cdcr specific properties must be defined in solrconfig.xml at the time of collection creation.
  - A non-cdcr collection cannot be converted later.
- Requires collection *'reload'* when cdcr properties are changed.
- Bootstrap takes significant time if source collection index is large. Index fetch is single threaded process.
- Doesn't support transaction and pull type replicas.

Forthcoming Improvements

- Moving configurations from solrconfig.xml to persist in zookeeper, per collection: */collections/<collection-name/cdcr-config.json*
- Collection level APIs for Cdcr:
    - RELOADCDCR, MODIFYCDCR, more to be added
    - /solr/collections?name=<collection_name>&action=<action>
- Support for transaction and pull type replicas

References

- Demonstration related materials: https://github.com/sarkaramrit2/cdcr-talk
- How SolrCloud works
- Apache Solr Offical CDCR Documentation Page
- Transaction logs and commit details
- Types of replicas in Apache Solr
- Relevant talks based on CDCR:
  - https://www.youtube.com/watch?v=fAvO8bHTh-Q
  - https://www.youtube.com/watch?v=83vbY9f3nXA

Thank you!