Apache Crail incubating Documentation

Release 1.0

–

Aug 21, 2022

OVERVIEW

Apache Crail (Incubating) is a high-performance distributed data store designed for fast sharing of ephemeral data in distributed data processing workloads.

CHAPTER

ONE

INTRODUCTION

Apache Crail (Incubating) is a fast multi-tiered distributed storage system designed from ground up for high-performance network and storage hardware. The unique features of Crail include:

- Zero-copy network access from userspace
- Integration of multiple storage tiers such DRAM, flash and disaggregated shared storage
- Ultra-low latencies for both meta data and data operations. For instance: opening, reading and closing a small file residing in the distributed DRAM tier less than 10 microseconds, which is in the same ballpark as some of the fastest RDMA-based key/value stores
- High-performance sequential read/write operations: For instance: read operations on large files residing in the distributed DRAM tier are typically limited only by the performance of the network
- Very low CPU consumption: a single core sharing both application and file system client can drive sequential read/write operations at the speed of up to 100Gbps and more
- Asynchronous API leveraging the asynchronous nature of RDMA-based networking hardware
- Extensible plugin architecture: new storage tiers tailored to specific hardware can be added easily

Crail is implemented in Java offering a Java API which integrates directly with the Java off-heap memory. Crail is designed for performance critical temporary data within a scope of a rack or two.

CHAPTER

TWO

DEPLOY CRAIL

Download the latest binary image from here and *configure* it. Alternatively you can *build from source* or use our *Docker container image*.

CHAPTER

THREE

BUILDING FROM SOURCE

Follow the steps below to build Crail from source.

3.1 Requirements

- Java 8 or higher
- RDMA-based network, e.g., Infiniband, iWARP, RoCE. There are two options to run Crail without RDMA networking hardware: (a) use SoftiWARP, (b) us the TCP/DRAM storage tier
- Libdisni.so, available as part of DiSNI

3.2 Building

To build Crail from source using Apache Maven execute the following steps:

1. (a) Clone from Apache: `git clone http://git-wip-us.apache.org/repos/asf/incubator-crail.git` or

   (b) Clone from Github: `git clone https://github.com/apache/incubator-crail` or

   (c) Download and unpack the latest source release from here
2. Run: `mvn -DskipTests install`
3. Copy tarball from `assembly/target` to the cluster and unpack it using `tar xvfz crail-X.Y-incubating-bin.tar.gz`

Note: *later, when deploying Crail, make sure libdisni.so is part of your LD_LIBRARY_PATH. The easiest way to make it work is to copy libdisni.so into $CRAIL_HOME/lib/*

CHAPTER FOUR

DOCKER

The easiest way to run Crail is to use Docker images. We provide two preconfigured Docker images:

(1) TCP/DRAM: **apache/crail**

(2) RDMA/DRAM: **apache/crail_rdma**

If you want to run other or more complex configurations you can use either image as a basis and provide your own configuration files. Refer to *Own configurations* for details.

(1) and 2. share the following configuration parameters:

Property	Default Value	Description
`NAMENODE_HOST`	localhost	Namenode hostname/ip to bind to
`NAMENODE_PORT`	9060	Namenode port to listen on
`INTERFACE`	eth0	Datanode network interface to listen on
`DATAPATH`	/dev/hugepages/data	Datanode hugepage path to data
`STORAGELIMIT`	1073741824	Datanode Size (Bytes) of DRAM to provide
`CACHEPATH`	/dev/hugepages/cache	Client/Datanode hugepage path to buffer cache
`CACHELIMIT`	0	Size (Bytes) of hugepage buffer cache

These properties can be specified as environment variables when starting a Docker image with `-e <property>=<value>`.

4.1 TCP image

To run Crail you first need to start the namenode. For example

```
docker run -it --network host -e NAMENODE_HOST=host02 -e INTERFACE=eth5 apache/crail␣
↪namenode
```

starts a TCP namenode using Docker's host network configuration on host02 on interface eth5. The TCP tier allows for other network configurations. Refer to https://docs.docker.com/network/ for details.

Once the namenode has started start a Crail TCP storage tier (preferably on a different node). For example

```
docker run -it --network host -e NAMENODE_HOST=host02 -e INTERFACE=eth5 apache/crail␣
↪datanode
```

starts a TCP datanode with 1GB of storage (default) listening on eth5. It is recommended to put the data directory on a hugetlb mount. You can do this by passing an mounted hugetlb directory on the host as a volume to Docker. For example

```
docker run -it --network host -e NAMENODE_HOST=host02 -e INTERFACE=eth5 -v /dev/
↪hugepages:/dev/hugepages apache/crail datanode
```

passes the hugetlb mount /dev/hugepages to the container.

4.2 RDMA image

To run Crail/RDMA/DRAM you can start the namenode as follows:

```
docker run -it --network host -e NAMENODE_HOST=host02 -e INTERFACE=eth5 --cap-add=IPC_
↪LOCK --device=/dev/infiniband/uverbs0 --device=/dev/infiniband/rdma_cm -v /dev/
↪hugepages:/dev/hugepages apache/crail_rdma namenode
```

This starts a namenode on host02 using the host's network on eth5. Note that the uverbs device needs to match the interface.

To run a RDMA storage tier:

```
docker run -it --network host -e NAMENODE_HOST=host02 -e INTERFACE=eth5 --cap-add=IPC_
↪LOCK --device=/dev/infiniband/uverbs0 --device=/dev/infiniband/rdma_cm -v /dev/
↪hugepages:/dev/hugepages apache/crail_rdma datanode
```

Note: *The RDMA docker image provides default RDMA provider libraries from Ubuntu 18.04. They might not be compatible with your host's RDMA stack. To install your own RDMA libraries use -v or create your own Docker image from crail_rdma.*

4.3 Own configurations

If you want to run more complex configurations that are not covered by the options above you have two options:

(1) Create your own Docker image by creating a Dockerfile and using the crail images as a source. You can then change the configuration by e.g. copying your own config into the image

(2) Pass your config as a volume with `-v <local_path>:<docker_path>`

CHAPTER

FIVE

CONFIGURATION

To configure Crail use the *.template files as a basis and modify it to match your environment. Set the `$CRAIL_HOME` environment variable to your Crail deployment's path.

```
cd $CRAIL_HOME/conf
mv crail-site.conf.template crail-site.conf
mv crail-env.sh.template crail-env.sh
mv core-site.xml.template core-site.xml
mv slaves.template slaves
```

Note: *Docker containers can be configured by using config files above. However it is only recommended for complex configurations. See Docker for details.*

The purpuse of each of these files are:

- *crail-site.conf*: Configuration of the file system, data tiers and RPC
- *crail-env.sh*: Allows to pass additional JVM arguments
- *core-site.xml*: Configuration of the HDFS adapter
- *slaves*: Used by the start-crail.sh script to ease running Crail on multiple machines

5.1 crail-site.conf

There are a general file system properties and specific properties for the different storage tiers. Typical properties you might want to change are:

Property	Default Value	Description
`crail.namenode.address`	crail://localhost:9060	Namenode hostname and port
`crail.cachelimit`	1073741824	Size (byte) of client buffer cache
`crail.cachepath`	/dev/hugepages/cache	Hugepage path to client buffer cache

Advanced properties (*Only modify if you know what you are doing*):

Property	Default Value	Description
`crail.directorydepth`	16	Maximum depth of directory tree
`crail.tokenexpiration`	10	Seconds write token is valid
`crail.blocksize`	1048576	Size (byte) of block
`crail.user`	crail	Username used for HDFS adapter
`crail.debug`	false	Enable debug output
`crail.statistics`	true	Collect statistics
`crail.rpctimeout`	1000	RPC timeout in milliseconds
`crail.datatimeout`	1000	Data operation timeout in milliseconds
`crail.buffersize`	1048576	Size (byte) of buffer (buffered stream)
`crail.slicesize`	524288	Size (byte) of slice (transfer unit)
`crail.singleton`	true	Only create a single instance of the FS
`crail.regionsize`	1073741824	Size (byte) of allocation unit (Cache)
`crail.directoryrecord`	512	Size (byte) of directory entry
`crail.directoryrandomize`	true	Randomize iteration of directories
`crail.cacheimpl`	org.apache.crail.memory.MappedBufferCache	Client buffer cache implementation
`crail.namenode.fileblocks`	16	File
`crail.namenode.blockselection`	roundrobin	Block selection algorithm: roundrobin or random

5.1.1 RPC

Crail's modular architecture allows to plugin different kinds of RPC implementations. The `crail.namenode.rpctype` property is used to configure the RPC implementation. We currently offer two implementations:

- A TCP implementation based on narpc (default): **org.apache.crail.namenode.rpc.tcp.TcpNameNode**
- A RDMA implementation based on darpc: **org.apache.crail.namenode.rpc.darpc.DaRPCNameNode**

Logging

To allow shutting down the namenode without loosing data Crail offers namenode logging. It can be enabled by setting a path to the log file with `crail.namenode.log`.

Note: *this feature is experimental and should be used with caution*

5.1.2 Storage Tiers

Crail offers multiple types of datanode dependent on your network and storage requirements:

(a) TCP storage tier backed by DRAM (default)

(b) RDMA storage tier backed by DRAM

(c) NVMe over Fabrics storage tier, typically backed by NVMe drives

Crail allows to use multiple storage tier types together, e.g. to store hot data on DRAM and cold data on NVMe, or extend your DRAM by NVMe storage. Storage types can be configured as a comma separated list by setting the `crail.storage.types` property:

(a) TCP: **org.apache.crail.storage.tcp.TcpStorageTier**

(b) RDMA: **org.apache.crail.storage.rdma.RdmaStorageTier**

(c) NVMf: **org.apache.crail.storage.nvmf.NvmfStorageTier**

Each of the storage types in the list defines a storage class, starting from storage class 0. Types can appear multiple times to allow defining multiple storage classes for a type. The maximum number of storage classes needs to be specified with the `crail.storage.classes` property (default = 1). In the default configuration storage classes are used in incremental order, i.e. storage class 0 is used until no more space is left then storage class 1 is used and so on. However filesystem nodes (e.g. files) can also be created on a particular storage class and can be configured to inherit the storage class of its container. The default storage class of `/` is 0 however it can be configured via `crail.storage.rootclass`.

Storage tiers send keep alive messages to the namenode to indicate that they are still running and no error has occured. The interval in which keep alive message are send can be configured in seconds with `crail.storage.keepalive`.

Some of the configuration properties can be set via the command line when starting a storage tier. Refer to *Run* for details.

TCP Tier

The TCP storage tier (org.apache.crail.storage.tcp.TcpStorageTier) is backed by DRAM. The following properties can be set to configure the storage tier:

Property	Default Value	Description
`crail.storage.tcp.interface`	eth0	Network interface to bind to
`crail.storage.tcp.storagelimit`	1073741824	Size (Bytes) of DRAM to provide, multiple of allocation size
`crail.storage.tcp.datapath`	/dev/hugepages/data	Hugepage path to data

Advanced properties:

Property	Default Value	Description
`crail.storage.tcp.port`	50020	Port to listen on
`crail.storage.tcp.allocationsize`	crail.regionsize	Allocation unit
`crail.storage.tcp.queuedepth`	16	Data operation queue depth (single connection)
`crail.storage.tcp.cores`	1	Threads to process requests

RDMA Tier

The RDMA storage tier (org.apache.crail.storage.rdma.RdmaStorageTier) is backed by DRAM. The following properties can be set to configure the storage tier:

Property	Default Value	Description
`crail.storage.rdma.interface`	eth0	Network interface to bind to
`crail.storage.rdma.storagelimit`	1073741824	Size (Bytes) of DRAM to provide; multiple of allocation size
`crail.storage.rdma.datapath`	/dev/hugepages/data	Hugepage path to data

Advanced properties:

Property	Default Value	Description
`crail.storage.rdma.port`	50020	Port to listen on
`crail.storage.rdma.allocationsize`	crail.regionsize	Allocation unit
`crail.storage.rdma.localmap`	true	Use mmap if client is colocated with data tier
`crail.storage.rdma.queuesize`	32	Data operation queue depth (single connection)
`crail.storage.rdma.type`	passive	Operation type: passive or active (see DiSNI)
`crail.storage.rdma.persistent`	false	Allow restarting a data tier if namenode logging is used
`crail.storage.rdma.backlog`	100	Listen backlog
`crail.storage.rdma.connecttimeout`	1000	Connect timeout in milliseconds

NVMf Tier

The NVMf storage tier (org.apache.crail.storage.nvmf.NvmfStorageTier) is typically backed by NVMe drives. However some target implementations support using any block device. Unlike the RDMA and TCP storage tier the NVMf storage tier is not involved in any data operation but only is used to provide metadata information. Crail uses the jNVMf library to connect to a standard NVMf target to gain metadata information about the storage and provide the information to the namenode. Clients directly connect to the NVMf target. Crail has been tested to run with the Linux kernel, SPDK and Mellanox ConnectX-5 offloading target.

The following properties can be set to configure the storage tier:

Property	Default Value	Description
`crail.storage.nvmf.ip`	localhost	IP/hostname of NVMf target
`crail.storage.nvmf.port`	50025	Port of NVMf target
`crail.storage.nvmf.nqn`	nqn.2017-06.io.crail:cnode	NVMe qualified name of NVMf controller
`crail.storage.nvmf.namespace`	1	Namespace of NVMe device
`crail.storage.nvmf.hostnqn`	<random 128bit UUID>	NVMe qualified name of host

Advanced properties:

Property	Default Value	Description
`crail.storage.nvmf.allocationsize`	crail.regionsize	Allocation unit
`crail.storage.nvmf.queueSize`	64	NVMf submission queue size
`crail.storage.nvmf.stagingcachesize`	262144	Staging cache size (byte) for read-modify-write operations

5.2 crail-env.sh

Modify crail-env.sh to pass additional JVM arguments to `crail` respectively `start-crail.sh`.

It is recommended to increase heap (e.g. `-Xmx24g`) and young generation heap size (e.g. `-Xmn16g`) for the namenodes and TCP datanodes to improve performance for large deployments.

5.3 core-site.xml

To configure the HDFS adapter modify core-site.xml. For example the Crail shell `crail fs` uses the HDFS adapter thus requiring the core-site.xml file to be setup. Modify `fs.defaultFS` to match `crail.namenode.address` in *crail-site.conf*. The default is:

```
<property>
  <name>fs.defaultFS</name>
  <value>crail://localhost:9060</value>
</property>
```

5.4 slaves

The slaves file can be used to ease starting Crail on larger deployments. Refer to *Run* for details. Each line should contain a hostname where a storage tier is supposed to be started. Make sure the hostname allows passwordless ssh connections. Note that the hostnames are not used by the storage tier itself but only by the start/stop-crail.sh scripts to start and stop storage tiers. IP/hostname of the storage tiers or any other configuration option are either passed by command line arguments or via *crail-site.conf*. Command line arguments can be configured in the slaves file following the hostname.

CHAPTER SIX

RUN

For all deployments, make sure the $CRAIL_HOME environment variable is set on each machine to point to the top level Crail directory.

6.1 Starting Crail manually

The simplest way to run Crail is to start it manually on just a handful nodes. You will need to start the Crail namenode, plus at least one datanode. To start the namenode execute the following command on the host that is configured to be the namenode:

```
$CRAIL_HOME/bin/crail namenode
```

To start a datanode run the following command on a host in the cluster (ideally this is a different physical machine than the one running the namenode):

```
$CRAIL_HOME/bin/crail datanode
```

Now you should have a small deployment up with just one datanode. In this case the datanode is of type TCP/DRAM, which is the default datnode. If you want to start a different storage tier you can do so by passing a specific storage tier type. You can find a list of supported storage tiers *here*. For example:

```
$CRAIL_HOME/bin/crail datanode -t org.apache.crail.storage.nvmf.NvmfStorageTier
```

starts the NVMf datanode. Note that configuration in *crail-site.conf* needs to have the specific properties set of this type of datanode, in order for this to work. Some storage tiers allow to set *configuration* properties on the command line which can be appended after -- to the command line, e.g.:

```
$CRAIL_HOME/bin/crail datanode -t org.apache.crail.storage.nvmf.NvmfStorageTier -- -a␣
↪192.168.0.2
```

Each storage tier instance can only belong to one storage class however the same storage tier type can belong to multiple storage classes. Refer to *Storage Tiers* for details. If there is only one storage class per type the storage class is picked by the order in which they appear in `crail.storage.types` (*crail-site.conf*). Use `-c <storage_class>` To start a storage tier in a specific storage class, e.g.:

```
$CRAIL_HOME/bin/crail datanode -t org.apache.crail.storage.nvmf.NvmfStorageTier -c 1
```

starts a NVMf storage tier in storage class 1 (storage classes start from 0).

6.2 Storage Tier Command Line

Command line arguments of the storage tiers override configuration properties in *crail-site.conf*. Refer to *crail-site.conf* for a detailed explanation of the properties and their default values.

6.2.1 TCP

Argument	crail-site.conf
`-p <port>`	`crail.storage.tcp.port`
`-c <cores>`	`crail.storage.tcp.cores`

6.2.2 RDMA

Argument	crail-site.conf
`-i <interface>`	`crail.storage.rdma.interface`
`-p <port>`	`crail.storage.rdma.port`
`-s`	`crail.storage.rdma.persistent`

6.2.3 NVMf

Argument	crail-site.conf/Description
`-a <ip/hostname>`	`crail.storage.nvmf.ip`
`-p <port>`	`crail.storage.nvmf.port`
`-nqn <nqn>`	`crail.storage.nvmf.nqn`
`-n <namespace_id>`	Namespace id to use (default 1)
`-hostnqn <nqn>`	`crail.storage.nvmf.hostnqn`

6.3 Larger deployments

To run larger deployments start Crail using

```
$CRAIL_HOME/bin/start-crail.sh
```

Similarly, Crail can be stopped by using

```
$CRAIL_HOME/bin/stop-crail.sh
```

For this to work include the list of machines to start datanodes in the *slaves* file. You can start multiple datanode of different types on the same host as follows:

```
host02
host02 -t org.apache.crail.storage.nvmf.NvmfStorageTier -- -a 192.168.0.2
host03
```

In this example, we are configuring a Crail cluster with 2 physical hosts but 3 datanodes and two different storage tiers.

6.4 Starting Crail in Docker

Refer to *Docker* for how to run Crail in a Docker container.

SHELL

Crail provides an implementation of the HDFS API thus allows interaction using the HDFS shell. For the HDFS adapter to work properly the *core-site.xml* needs to be configured properly.

```
$CRAIL_HOME/bin/crail fs
```

Not all shell commands are support but the following operations have been tested to work:

```
$CRAIL_HOME/bin/crail fs -ls <crail_path>
$CRAIL_HOME/bin/crail fs -mkdir <crail_path>
$CRAIL_HOME/bin/crail fs -copyFromLocal <local_path> <crail_path>
$CRAIL_HOME/bin/crail fs -copyToLocal <crail_path> <local_path>
$CRAIL_HOME/bin/crail fs -cat <crail_path>
```

CHAPTER EIGHT

IOBENCH

The iobench tool allows to perform microbenchmarks on Crail.

8.1 Examples

Synchronously write 1MB 1024 times to get a 1GB file:

```
$CRAIL_HOME/bin/crail iobench -t write -f /filename -s $((1024*1024)) -k 1024
```

Read 1024 1MB buffers asynchronously with a batch size of 4:

```
$CRAIL_HOME/bin/crail iobench -t readSequentialAsync -f /filename -s $((1024*1024)) -k␣
↪1024 -b 4
```

8.2 Command Reference

Argument	Default	Experiment type	Description
`-t <experiment>`	-	N/A	• `write` - sequential sync write • `writeAsync` - sequential async write • `readSequential` - sequential sync read • `readRandom` - random sync read • `readSequentialAsync` - sequential async read • `readMultiStream` - multistream read • `createFile` - create file RPC • `createFileAsync` - create file async RPC • `createMultiFile` - create multifile • `getKey` - getKey RPC • `getFile` - getFile sync RPC • `getFileAsync` - getFile async RPC • `enumerateDir` - enumerate directory • `browseDir` - browse directory • `writeInt` - write integer • `readInt` - read integer • `seekInt` - seek integer • `readMultiStreamInt` - read integer multistream • `printLocationclass` - print machine's location class
`-f <path>`	/tmp.dat	• `write` • `writeAsync` • `readSequential` • `readRandom` • `readSequentialAsync` • `readMultiStream` • `createFile` • `createFileAsync`	Path to perform operation with

CHAPTER

NINE

FSCK

The fsck is used to query Crail internals and perform management operations.

9.1 Reference

Argument	Default	Experiment type	Description
`-t <experiment>`	-	N/A	• `getLocations` • `directoryDump` • `namenodeDump` • `blockStatistics` • `ping` • `createDirectory`
`-f <path>`	/tmp.dat	• `getLocations` • `directoryDump` • `namenodeDump` • `blockStatistics` • `createDirectory`	Path to perform operation with
`-y <offset>`	0	• `getLocations`	Offset into file
`-l <length>`	1	• `getLocations`	Length starting from offset (-y)
`-c <storage_class>`	0	• `createDirectory`	Storage class of directory
`-p <location_class>`	0	• `createDirectory`	Location class of directory

CHAPTER

TEN

SPARK

Crail can be used to increase performance or enhance flexibility in Apache Spark. We provide multiple plugins to allow Crail to be used as:

- *HDFS Adapter*: input and output
- *Spark-IO*: shuffle data and broadcast store

10.1 HDFS Adapter

The Crail HDFS adapter is provided with every Crail *deployment*. The HDFS adpater allows to replace every HDFS path with a path on Crail. However for it to be used for input and output in Spark the jar file paths have to be added to the Spark configuration spark-defaults.conf:

```
spark.driver.extraClassPath      $CRAIL_HOME/jars/*
spark.executor.extraClassPath    $CRAIL_HOME/jars/*
```

Data in Crail can be accessed by prepending the value of `crail.namenode.address` from *crail-site.conf* to any HDFS path. For example `crail://localhost:9060/test` accesses `/test` in Crail. Note that Crail works independent of HDFS and does not interact with HDFS in any way. However Crail does not completely replace HDFS since we do not offer durability and fault tolerance cf. *Introduction*. A good fit for Crail is for example inter-job data that can be recomputed from the original data in HDFS.

10.2 Spark-IO

Crail-Spark-IO contains various I/O accleration plugins for Spark tailored to high-performance network and storage hardware (RDMA, NVMef, etc.). Spark-IO is not provided with the default Crail deployment but can be obtained here. Spark-IO currently contains two IO plugins: a shuffle engine and a broadcast module. Both plugins inherit all the benefits of Crail such as very high performance (throughput and latency) and multi-tiering (e.g., DRAM and flash).

10.2.1 Requirements

- Spark >= 2.0
- Java 8
- Maven
- Crail >= 1.0

10.2.2 Building

To build Crail execute the following steps:

1. Obtain a copy of Crail-Spark-IO from Github
2. Make sure your local maven repository contains Crail, if not build Crail from *source*
3. Run: `mvn -DskipTests install`

10.2.3 Configure Spark

To configure the crail shuffle plugin add the following lines to spark-defaults.conf

```
spark.shuffle.manager           org.apache.spark.shuffle.crail.CrailShuffleManager

spark.driver.extraClassPath     $CRAIL_HOME/jars/*:<path>/crail-spark-X.Y.jar:.
spark.executor.extraClassPath   $CRAIL_HOME/jars/*:<path>/crail-spark-X.Y.jar:.
```

Since Spark version 2.0.0, broadcast is no longer an exchangeable plugin, unfortunately. To use the Crail broadcast plugin in Spark it has to be manually added to Spark's BroadcastManager.scala.

10.3 Crail-TeraSort

10.4 SQL

CHAPTER

ELEVEN

PROGRAMMING AGAINST CRAIL

The best way to program against Crail is to use Maven. Make sure you have the Crail dependency specified in your application pom.xml file with the latest Crail version (e.g. 1.1-incubating):

```
<dependency>
  <groupId>org.apache.crail</groupId>
  <artifactId>crail-client</artifactId>
  <version>X.Y</version>
</dependency>
```

Then, create a Crail client as follows:

```
CrailConfiguration conf = new CrailConfiguration();
CrailStore store = CrailStore.newInstance(conf);
```

Make sure the `$CRAIL_HOME/conf` directory is part of the classpath.

Crail supports different file types. The simplest way to create a file in Crail is as follows:

```
CrailFile file = store.create(filename, CrailNodeType.DATAFILE, CrailStorageClass.
↪DEFAULT, CrailLocationClass.DEFAULT).get().syncDir();
```

Aside from the actual filename, the `create()` call takes as input the storage and location classes which are preferences for the storage tier and physical location that this file should be created in. Crail tries to satisfy these preferences later when the file is written. In the example we do not request any particular storage or location affinity.

This `create()` command is non-blocking, calling `get()` on the returning future object awaits the completion of the call. At that time, the file has been created, but its directory entry may not be visible. Therefore, the file may not yet show up in a file enumeration of the given parent directory. Calling `syncDir()` waits to for the directory entry to be completed. Both the `get()` and the `syncDir()` operation can be deffered to a later time at which they may become non-blocking operations.

Once the file is created, a file stream can be obtained for writing:

```
CrailBufferedOutputStream outstream = file.getBufferedOutputStream(1024);
```

Here, we create a buffered stream so that we can pass heap byte arrays as well. We could also create a non-buffered stream using

```
CrailOutputStream outstream = file.getDirectOutputStream(1024);
```

In both cases, we pass a write hint (1024 in the example) that indicates to Crail how much data we are intending to write. This allows Crail to optimize metadatanode lookups. Crail never prefetches data, but it may fetch the metadata of the very next operation concurrently with the current data operation if the write hint allows to do so.

Once the stream has been obtained, there exist various ways to write a file. The code snippet below shows the use of the asynchronous interface:

```
CrailBuffer dataBuf = fs.allocateBuffer();
Future<DataResult> future = outputStream.write(dataBuf);
...
future.get();
```

Reading files works very similar to writing. There exist various examples in org.apache.crail.tools.CrailBenchmark.

CHAPTER

TWELVE

HOW TO RELEASE

This guide explains how to prepare for a source and binary release of Apache Crail (Incubating) project for a release number of `x.y` (indicated as `${RELEASE_VERSION}`) and release candidate number `X` as `rcX` (indicated as `${RELEASE_CANDIDATE}`).

Table of Contents

12.1 1. Configure your environment for a release

Before we do a release, lets start by setting up the release environment (and cross check some of the other settings).

12.1.1 1.1 Setup git username

Make sure `git` is configured properly.

```
git config user.email "your_id@apache.org"
git config user.name "your_name"
```

12.1.2 1.2 Setup keys

1. Generate a code signing key, https://www.apache.org/dev/openpgp.html#generate-key

```
gpg --gen-key
```

2. Check the preference for SHA-1 for your key, https://www.apache.org/dev/openpgp.html#key-gen-avoid-sha1

```
gpg --edit-key your_key_id
```

3. Upload/publish the key: https://www.apache.org/dev/release-signing.html#keyserver-upload

```
gpg --keyserver pgp.mit.edu --send-keys <key id>
```

4. Add your KEY in the KEYS file:

```
svn co https://dist.apache.org/repos/dist/release/incubator/crail/
cd crail
(gpg --list-sigs <key id> && gpg --armor --export <key id>) >> KEYS
svn commit KEYS -m "your_name (id@apache.org) keys"
```

6. Update your profile https://id.apache.org/ with the fingerprint of the key. Find your fingerprint at

```
gpg --fingerprint
```

12.1.3 1.3 Maven Settings File

Prior to performing an Apache Crail release, you must have an entry such as this in your ~/.m2/settings.xml file to authenticate when deploying the release artifacts.

```
 <?xml version="1.0" encoding="UTF-8"?>
 <settings xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.
↪apache.org/xsd/settings-1.0.0.xsd"
     xmlns="http://maven.apache.org/SETTINGS/1.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <servers>
     <server>
       <id>apache.snapshots.https</id>
       <username>USERNAME</username>
       <password>PASSWORD</password>
     </server>
     <server>
       <id>apache.releases.https</id>
       <username>USERNAME</username>
       <password>PASSWORD</password>
     </server>
   </servers>
</settings>
```

How to put encrypted password https://maven.apache.org/guides/mini/guide-encryption.html

12.2 2. Preparing for a release

A release consists of a doing a (i) source release; (b) binary release; (iii) uploading maven artifacts; (iv) updating documentation. To do a version release of `x.y` (which is referred to as `${RELEASE_VERSION}`), follow these steps:

1. Update `GIT_COMMIT` in `docker/Dockerfile` to the newest release tag, e.g. `v1.2`. In `docker/RDMA/Dockerfile` update `FROM` to the last Crail version as defined in the parent Dockerfile e.g. `1.2` (without "v") and `DISNI_COMMIT` to the DiSNI version as specified in the parent pom file.
2. Go through the closed JIRAs and merge requests, and update the HISTORY.md file about what is new in the new release version.
3. Perform `mvn apache-rat:check` and make sure it is a SUCCESS.
4. Perform `mvn checkstyle:check`. For now it will fail, but make sure that it runs. We need to gradually fix it. [JIRA-59](https://issues.apache.org/jira/browse/CRAIL-59)
5. Perform maven prepare release in the interactive mode.

```
mvn release:prepare -P apache-release -Darguments="-DskipTests"  -DinteractiveMode=true -
↪Dresume=false
```

The interactive mode allows us to explicitly name the current release version, release candidate, and next version. The convention here is to follow `apache-crail-${RELEASE_VERSION}-incubating-${RELEASE_CANDIDATE}` naming, starting from release candidate 0. So, for a ${RELEASE_VERSION} of 2.12 and release candidate 10, the name would be `apache-crail-2.12-incubating-rc10`. For `rc0`, we let the command increment the pom version. Here is an example run of this command for the release for `1.2-incubating`. As you can see, the first time you run the command (for `rc0`, the version are picked automatically). For subsequent RCs, you have to make sure that version is not incremented unless a RC is successfully voted on. Between RCs, we expect everything to remain the same except the `SCM release tag` that you must keep in sync with the release candidate.

NOTE: the SCM tag does not have `incubating` in its name, and uses a `v` prefix.

```
[INFO] Checking dependencies and plugins for snapshots ...
What is the release version for "Crail Project Parent POM"? (org.apache.crail:crail-
↪parent) 1.2-incubating: : 1.2-incubating
What is the release version for "Crail Client Project"? (org.apache.crail:crail-client)␣
↪1.2-incubating: : 1.2-incubating
What is the release version for "Crail RPC Project"? (org.apache.crail:crail-rpc) 1.2-
↪incubating: : 1.2-incubating
What is the release version for "Crail Namenode Project"? (org.apache.crail:crail-
↪namenode) 1.2-incubating: : 1.2-incubating
What is the release version for "Crail Storage Project"? (org.apache.crail:crail-
↪storage) 1.2-incubating: : 1.2-incubating
What is the release version for "Crail RDMA Project"? (org.apache.crail:crail-storage-
↪rdma) 1.2-incubating: : 1.2-incubating
What is the release version for "Crail NVMf Project"? (org.apache.crail:crail-storage-
↪nvmf) 1.2-incubating: : 1.2-incubating
What is the release version for "Crail Storage NaRPC Project"? (org.apache.crail:crail-
↪storage-narpc) 1.2-incubating: : 1.2-incubating
What is the release version for "Crail DaRPC Project"? (org.apache.crail:crail-rpc-
↪darpc) 1.2-incubating: : 1.2-incubating
What is the release version for "Crail RPC/TCP Project"? (org.apache.crail:crail-rpc-
↪narpc) 1.2-incubating: : 1.2-incubating
What is the release version for "Crail HDFS Project"? (org.apache.crail:crail-hdfs) 1.2-
↪incubating: : 1.2-incubating
```

(continues on next page)

(continued from previous page)

```
What is the release version for "Crail Project Assembly"? (org.apache.crail:crail-
↪assembly) 1.2-incubating: : 1.2-incubating
What is SCM release tag or label for "Crail Project Parent POM"? (org.apache.crail:crail-
↪parent) crail-parent-1.2-incubating: : v1.2-rc0
What is the new development version for "Crail Project Parent POM"? (org.apache.
↪crail:crail-parent) 1.3-incubating-SNAPSHOT: : 1.3-incubating-SNAPSHOT
What is the new development version for "Crail Client Project"? (org.apache.crail:crail-
↪client) 1.3-incubating-SNAPSHOT: : 1.3-incubating-SNAPSHOT
What is the new development version for "Crail RPC Project"? (org.apache.crail:crail-
↪rpc) 1.3-incubating-SNAPSHOT: : 1.3-incubating-SNAPSHOT
What is the new development version for "Crail Namenode Project"? (org.apache.
↪crail:crail-namenode) 1.3-incubating-SNAPSHOT: : 1.3-incubating-SNAPSHOT
What is the new development version for "Crail Storage Project"? (org.apache.crail:crail-
↪storage) 1.3-incubating-SNAPSHOT: : 1.3-incubating-SNAPSHOT
What is the new development version for "Crail RDMA Project"? (org.apache.crail:crail-
↪storage-rdma) 1.3-incubating-SNAPSHOT: : 1.3-incubating-SNAPSHOT
What is the new development version for "Crail NVMf Project"? (org.apache.crail:crail-
↪storage-nvmf) 1.3-incubating-SNAPSHOT: : 1.3-incubating-SNAPSHOT
What is the new development version for "Crail Storage NaRPC Project"? (org.apache.
↪crail:crail-storage-narpc) 1.3-incubating-SNAPSHOT: : 1.3-incubating-SNAPSHOT
What is the new development version for "Crail DaRPC Project"? (org.apache.crail:crail-
↪rpc-darpc) 1.3-incubating-SNAPSHOT: : 1.3-incubating-SNAPSHOT
What is the new development version for "Crail RPC/TCP Project"? (org.apache.crail:crail-
↪rpc-narpc) 1.3-incubating-SNAPSHOT: : 1.3-incubating-SNAPSHOT
What is the new development version for "Crail HDFS Project"? (org.apache.crail:crail-
↪hdfs) 1.3-incubating-SNAPSHOT: : 1.3-incubating-SNAPSHOT
What is the new development version for "Crail Project Assembly"? (org.apache.
↪crail:crail-assembly) 1.3-incubating-SNAPSHOT: : 1.3-incubating-SNAPSHOT
[INFO] Transforming 'Crail Project Parent POM'...
[...]
```

In case, if you are not sure about some setting, try *-DdryRun=true*. If something goes wrong then `mvn release:rollback`.

NOTE: the binary file and associated signature (asc) and sha512 files are generated at `assembly/target/crail-${RELEASE_VERSION}-incubating-bin.tar.gz`. The source file and associated signature (asc) and sha512 files are at `assembly/target/crail-${RELEASE_VERSION}-incubating-src.tar.gz`.

6. We need to upload the generated artifacts to the "Stage" SVN at https://dist.apache.org/repos/dist/dev/incubator/crail/. So lets prepare that in a SVN staging directory (SSD)

```
svn co https://dist.apache.org/repos/dist/dev/incubator/crail/
cd crail
mkdir ${RELEASE_VERSION}-${RELEASE_CANDIDATE}
# lets call the created directory the svn staging directory (SSD)
SSD=`pwd`/${RELEASE_VERSION}-${RELEASE_CANDIDATE}
```

7. Collect all artifacts to release in the SVN staging directory (SSD)

```
# copy files from the crail build location to the SVN staging directory (SSD)
# binary file
cp assembly/target/apache-crail-${RELEASE_VERSION}-incubating-bin.tar.gz ${SSD}/
# source file
```

(continues on next page)

(continued from previous page)

```
cp assembly/target/apache-crail-${RELEASE_VERSION}-incubating-src.tar.gz ${SSD}/
# copy signature files
cp assembly/target/apache-crail-${RELEASE_VERSION}-incubating-bin.tar.gz.asc ${SSD}/
cp assembly/target/apache-crail-${RELEASE_VERSION}-incubating-src.tar.gz.asc ${SSD}/
# copy checksum files
cp assembly/target/apache-crail-${RELEASE_VERSION}-incubating-bin.tar.gz.sha512 ${SSD}/
cp assembly/target/apache-crail-${RELEASE_VERSION}-incubating-src.tar.gz.sha512 ${SSD}/
# step in the SVN staging directory
cd ${SSD}
```

8. Verify the checksums for source and binary files

```
sha512sum -c apache-crail-${RELEASE_VERSION}-incubating-src.tar.gz.sha512
sha512sum -c apache-crail-${RELEASE_VERSION}-incubating-bin.tar.gz.sha512
```

9. Verify the signatures for source and binary files

```
gpg --verify apache-crail-${RELEASE_VERSION}-incubating-src.tar.gz.asc apache-crail-$
↪{RELEASE_VERSION}-incubating-src.tar.gz
gpg --verify apache-crail-${RELEASE_VERSION}-incubating-bin.tar.gz.asc apache-crail-$
↪{RELEASE_VERSION}-incubating-bin.tar.gz
```

10. Commit the files after verification in the SVN staging directory

```
svn add ${RELEASE_VERSION}-${RELEASE_CANDIDATE}
svn commit ${RELEASE_VERSION}-${RELEASE_CANDIDATE} -m "${RELEASE_VERSION}-${RELEASE_
↪CANDIDATE} release files"
```

11. Upload the artifacts to the Nexus https://repository.apache.org/index.html#welcome (login using your Apache ID) by calling

```
mvn release:perform -P apache-release  -Darguments="-DskipTests"
```

12. After upload you need to
    1. Close the staging repository at https://repository.apache.org
    2. Login to https://repository.apache.org.
    3. Go to "Staging Repos".
    4. Find the "orgapachecrail" repo with the Crail release. Be sure to expand the contents of the repo to confirm that it contains the correct Crail artifacts.
    5. Click on the "Close" button at top, and enter a brief description, such as "Apache Crail (Incubating) ${RELEASE_VERSION} release". **Note** this might fail on the very first attempt just repeat closing it.
    6. Copy the staging URL like `https://repository.apache.org/content/repositories/orgapachecrail-1000/`

13. [Optionally] Check if docker images have been created successfully https://hub.docker.com/r/apache/incubator-crail/ and https://hub.docker.com/r/apache/incubator-crail-rdma/. Make sure that the docker configuration file at https://github.com/apache/incubator-crail/blob/v${RELEASE_VERSION}-${RELEASE_CANDIDATE}/docker/RDMA/Dockerfile contains the right tag version for `FROM crail:[RELEASE_TAG]` and the right DiSNI version (which matches the pom file for this release) at `ARG DISNI_COMMIT="[DISNI_VERSION_FROM_CRAIL_POM]"`.

12.3 3. Voting on an RC

The voting is a 2 step process.

12.3.1 3.1 PPMC voting

First, we need to gather 3 binding votes (PPMC members) on the crail mailing list. To call the vote, you can use this template:

```
Subject: [VOTE] Release of Apache Crail-${RELEASE_VERSION}-incubating [${RELEASE_
↪CANDIDATE}]
============================================================================

Hi all,

This is a call for a vote on releasing Apache Crail ${RELEASE_VERSION}-incubating,␣
↪release candidate X.

The source and binary tarball, including signatures, digests, etc. can be found at:
https://dist.apache.org/repos/dist/dev/incubator/crail/${RELEASE_VERSION}-incubating-$
↪{RELEASE_CANDIDATE}/

The commit to be voted upon:
https://git-wip-us.apache.org/repos/asf?p=incubator-crail.git;a=commit;h=[REF]

The Nexus Staging URL:
https://repository.apache.org/content/repositories/orgapachecrail-[STAGE_ID]

Release artifacts are signed with the following key:
https://www.apache.org/dist/incubator/crail/KEYS

For information about the contents of this release, see:
https://git-wip-us.apache.org/repos/asf?p=incubator-crail.git;a=blob;f=HISTORY.md;h=$
↪{RELEASE_HASH}
or https://github.com/apache/incubator-crail/blob/v${RELEASE_VERSION}-${RELEASE_
↪CANDIDATE}/HISTORY.md

Please vote on releasing this package as Apache Crail ${RELEASE_VERSION}-incubating

The vote will be open for 72 hours.

[ ] +1 Release this package as Apache Crail ${RELEASE_VERSION}-incubating
[ ] +0 no opinion
[ ] -1 Do not release this package because ...


Thanks,
[YOUR_NAME]
```

Make sure that you modify (i) ${RELEASE_VERSION} in the subject and body; (ii) ${RELEASE_CANDIDATE} tags; (iii) ${RELEASE_HASH}; (iv) [STAGE_ID]; (iv) YOUR_NAME

After a successful vote, announce the result on the Crail mailing list:

```
Subject: [RESULT][VOTE] Crail v${RELEASE_VERSION}-${RELEASE_CANDIDATE} release
===============================================

Hi all,

Thanks for all who voted. I'm closing the vote since the 72 hours have passed. Here are␣
↪the results:
X + votes
Y - votes

I will call for the IPMC vote.

Thanks,
[YOUR_NAME]
```

12.3.2 3.2 IPMC voting

After a succesfull PPMC vote, we need to call for the IPMC vote on the `general@incubator.apache.org` (https://incubator.apache.org/guides/lists.html). You can use this template:

```
Subject:[VOTE] Apache Crail ${RELEASE_VERSION}-incubating (${RELEASE_CANDIDATE})
=================================================

Please vote to approve the source release of Apache Crail ${RELEASE_VERSION}-incubating (
↪${RELEASE_CANDIDATE}).
[If any] This release candidate fixes all issues raised in the last IPMC vote:
- x
- y
- z

The podling dev vote thread:

https://www.mail-archive.com/dev@crail.apache.org/???.html

The result:

https://www.mail-archive.com/dev@crail.apache.org/???.html

Commit hash: ${RELEASE_HASH}

https://git1-us-west.apache.org/repos/asf?p=incubator-crail.git;a=commit;h=${RELEASE_
↪HASH}

Release files can be found at:
https://dist.apache.org/repos/dist/dev/incubator/crail/${RELEASE_VERSION}-${RELEASE_
↪CANDIDATE}/

The Nexus Staging URL:
https://repository.apache.org/content/repositories/orgapachecrail-[STAGE_ID]

Release artifacts are signed with the following key:
```

(continues on next page)

(continued from previous page)

```
https://www.apache.org/dist/incubator/crail/KEYS

For information about the contents of this release, see:
https://git-wip-us.apache.org/repos/asf?p=incubator-crail.git;a=blob;f=HISTORY.md;h=$
↪{RELEASE_HASH}
or https://github.com/apache/incubator-crail/blob/v${RELEASE_VERSION}-${RELEASE_
↪CANDIDATE}/HISTORY.md

The vote is open for at least 72 hours and passes if a majority of at least 3 +1 PMC␣
↪votes are cast.

[ ] +1 Release this package as Apache Crail 1.0-incubating
[ ] -1 Do not release this package because ...

Thanks,
[YOUR_NAME]
```

After a successful vote, annouce the result as:

```
Subject: [RESULT][VOTE] Apache Crail ${RELEASE_VERSION}-incubating (${RELEASE_CANDIDATE})
=========================================================

Hi all,

Thanks for all your votes. Here is the result:
x + votes
y - votes

[If any] Some comments for future votes that I'm about to address:
- x
- y
- z

I'm going to release Crail ${RELEASE_VERSION}-incubating. Thank you all for making this␣
↪happen!

Thanks,
[YOUR_NAME]
```

Obviosuly not all calls to vote can succeed. In case of a failed vote, announce as:

```
Subject:[CANCEL][VOTE] Release of Apache Crail ${RELEASE_VERSION}-incubating (${RELEASE_
↪CANDIDATE})
===================================================================

Hi all,
I'm canceling the vote for Apache Crail ${RELEASE_VERSION}-incubating (${RELEASE_
↪CANDIDATE}), due to found/discussed issues.

I will prepare a new release candidate.

Thanks,
[YOUR_NAME]
```

NOTE: If your PPMC vote fails you have to redo the IPMC vote again after fixing the issues raised in the PPMC vote.

12.4 4. After acceptance

1. Tag the commit (on which the vote happened) with the release version without `-${RELEASE_CANDIDATE}`. So, for example, after a successful vote on `v1.2-rc5`, the hash will be tagged again with `v1.2` only.
2. Upload to the "release" (this is different from the "staging" SVN that we used before) SVN https://dist.apache.org/repos/dist/release/incubator

```
svn co https://dist.apache.org/repos/dist/release/incubator
cd incubator/crail
mkdir ${RELEASE_VERSION}-incubating
cd ${RELEASE_VERSION}-incubating
# copy the tar.gz. asc. and sha512 files for the src and binary releases
# Remove old releases and commit
```

3. Release nexus artifacts. Follow the step 12 in the release process but this time press the `release` button.

4. Write an announement email. You have to make announcement at two places, the general Apache announcement as well to crail mailing list. You can use this template to make the announcement:

```
Subject: [ANNOUNCE] Apache Crail ${RELEASE_VERSION}-incubating released
========================================================

The Apache Crail community is pleased to announce the release of
Apache Crail version ${RELEASE_VERSION}-incubating.

[If any] The key features of this release are:
- x
- y
- z

Crail is a high-performance distributed data store designed for fast
sharing of ephemeral data in distributed data processing workloads. You
can read more about Crail on the website: https://crail.apache.org/

The release is available at:
https://crail.incubator.apache.org/download/

The full change log is available here:
https://github.com/apache/incubator-crail/blob/v${RELEASE_VERSION}/HISTORY.md

We welcome any help and feedback. Check out https://crail.incubator.apache.org/community/
to get involved.

Thanks to all involved for making this first release happen!

Thanks,
[YOUR_NAME]

--
Apache Crail is an effort undergoing incubation at The Apache Software
```

(continues on next page)

(continued from previous page)

```
Foundation (ASF), sponsored by the Apache Incubator PMC. Incubation is
required of all newly accepted projects until a further review
indicates that the infrastructure, communications, and decision making
process have stabilized in a manner consistent with other successful
ASF projects. While incubation status is not necessarily a reflection
of the completeness or stability of the code, it does indicate that the
project has yet to be fully endorsed by the ASF.```
```

The Apache annoucement list is at `announce@apache.org`. You need to subscribe first.

5. Update the download page on the website
6. Social media (Twitter, LinkedIn announcements)
7. [Optionally] Check if docker images have been created successfully https://hub.docker.com/r/apache/incubator-crail/ and https://hub.docker.com/r/apache/incubator-crail-rdma/ with the new release tag.

12.5 5. Useful links

1. General info for release signing: https://www.apache.org/dev/release-signing.html
2. http://tephra.incubator.apache.org/ReleaseGuide.html
3. https://dubbo.incubator.apache.org/en-us/blog/prepare-an-apache-release.html

CHAPTER

THIRTEEN

CONTRIBUTE

For any potential changes/proposals we recommend that you open a JIRA ticket to have a discussion. After making necessary code changes, please open a pull request at Github, and update the JIRA.

CHAPTER FOURTEEN

CONTACT

Feel free to ask questions any questions on our mailing list: dev@crail.apache.org

If you find any issues please report them at https://issues.apache.org/jira/projects/CRAIL/issues