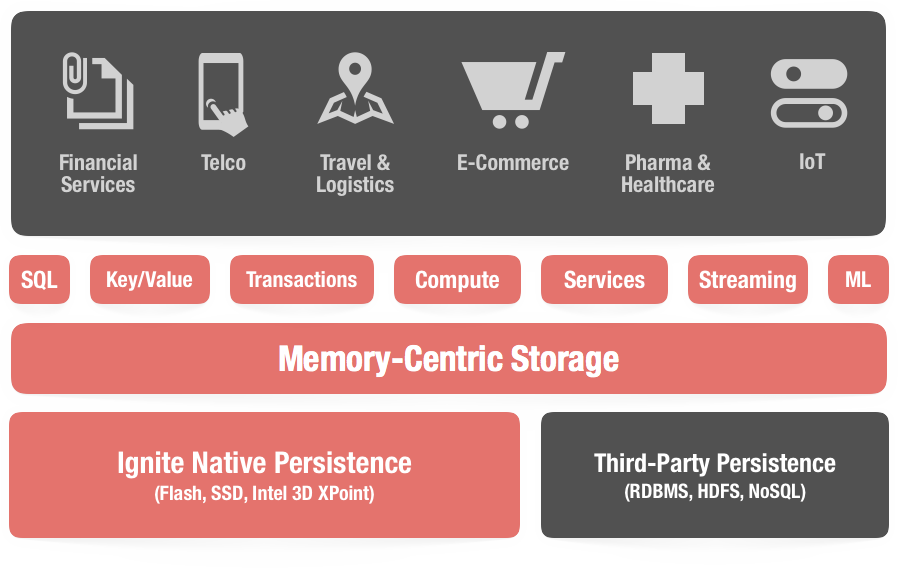
Apache Ignite

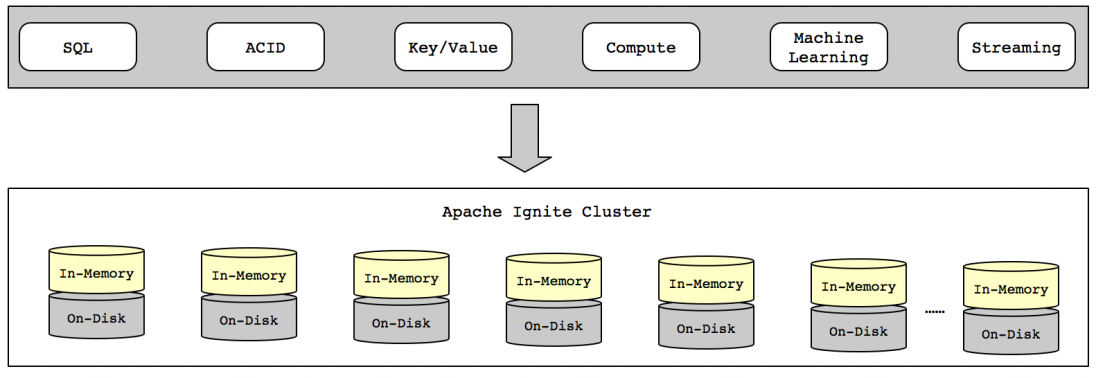
Apache Ignite是大数据系统中的内存分布式管理系统，其是一个高性能、集成和分布式的内存计算和事务平台，用于大规模的数据集处理，比传统的基于磁盘或闪存的技术具有更高的性能，同时它还为应用和不同的数据源之间提供高性能、分布式内存中数据组织管理的功能



Ignite是一个全功能的数据网格，可以用于纯内存模块，也可以用于带有Ignite的原生持久化，其可以和任何第三方的数据库集成，包含RDBMS和NoSQL。比如在Hadoop的HDFS、Kafka等，开发基于大数据平台下的SQL引擎，来操作HDFS、Kafak这类大数据存储介质。

# **Apache Ignite架构**

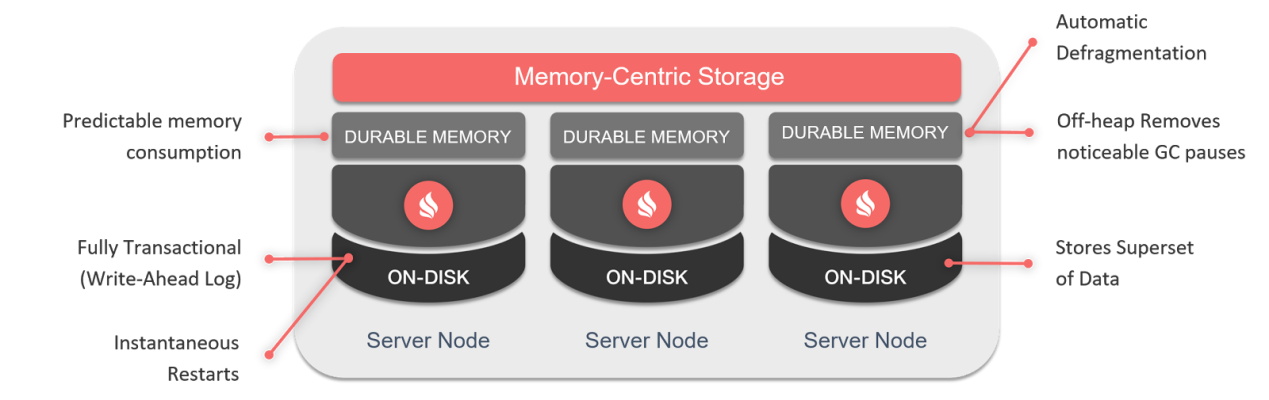
Apache Ignite是一个以内存为中心的分布式数据库、缓存和处理平台，可以在PB级数据中，以内存级的速度进行事务性、分析性以及流式负载的处理。其结构分布图如下所示：



Apache Ignite具有强一致性、高可用、强大的SQL、KV以及对应的应用接口。在整个Ignite集群的多个节点中，Ignite内存的数据模式有三种：LOCAL、REPLICATED和PARTITIONED，这增加了Ignite的扩展性，Ignite可以自动化的控制数据如何分区，使用者可以自动插入自定义的方法或者为了提高效率将部分数据存在一起。

## **固化内存**

Apache Ignite是以内存为中心的数据平台，其基于固化内存架构（Durable Memory），当其开启持久化存储特性时（Native Persistence），其可以在内存和磁盘中存储和处理数据集及索引，其架构图如下所示：

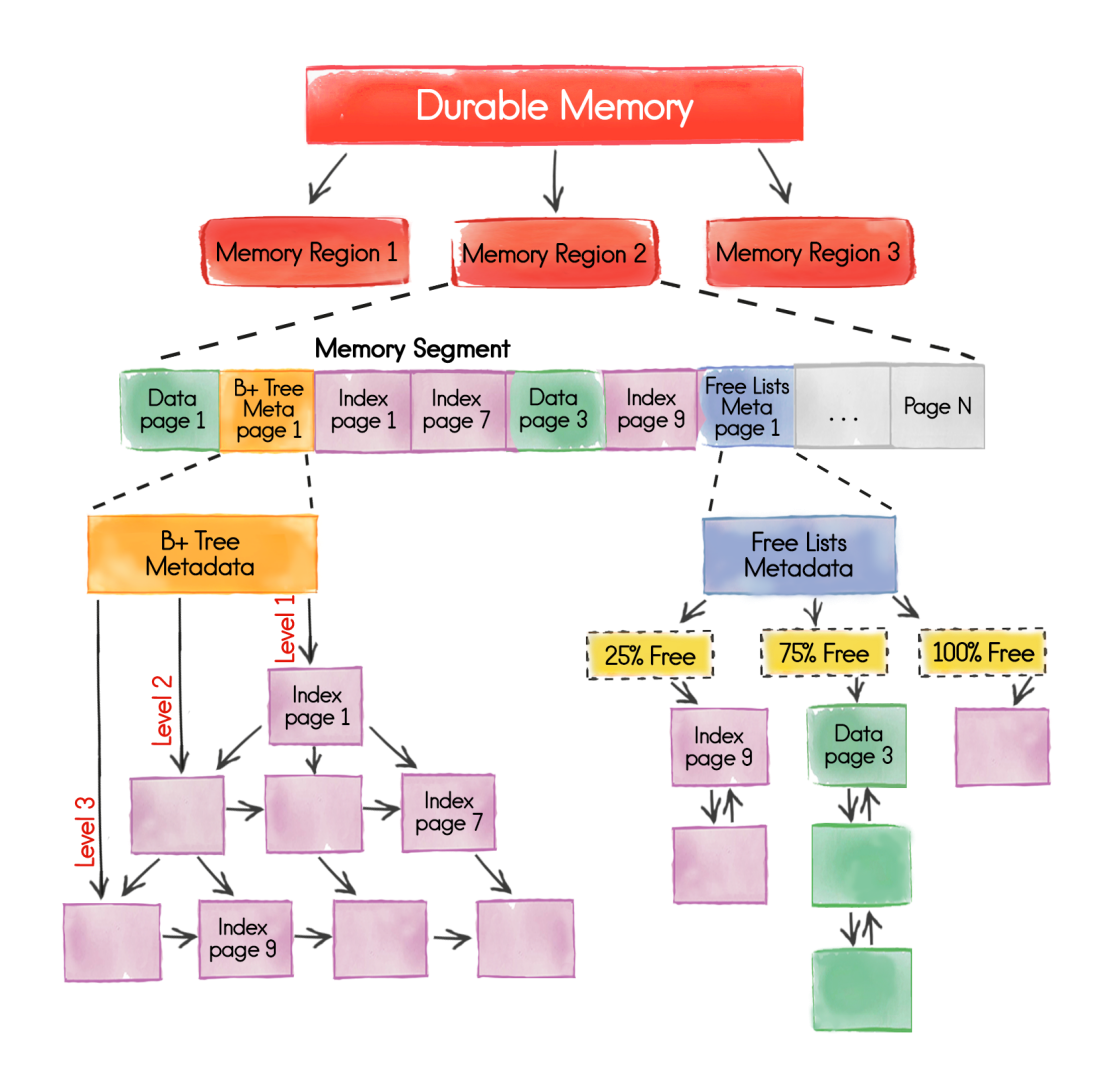
****

使用固化内存可以充分利用集群资源，包括内存的性能及磁盘的持久化功能。固化内存的功能类似于操作系统的虚拟内存，但是当启用持久化功能时，将磁盘作为内存数据的父集，从磁盘中读取内存中查询不到的数据。

Ignite的固化内存组件不仅将内存作为一个缓冲层，还视为一个全功能的存储层：当持久化关闭Ignite可以作为分布式内存数据库或者内存数据网格，持久化打开则Ignite成为一个分布式、可水平扩展的数据库，会保证完整的数据一致性以及集群故障的可恢复能力。这种架构以内存为中心，所有的处理在内存中处理，使用内存架构的特性和优势：

* 堆外内存： 所有的数据和索引都位于Java堆外，这样就可以轻易的处理集群内PB级别数据
* 消除明显的由垃圾回收导致的暂停： 所有的数据和索引都存储在堆外，所以应用是有可能导致长时间暂停的唯一来源
* 内存使用可预测，固化内存默认会使用尽可能多的内存和磁盘空间，但是也可以配置内存使用量来满足应用的需求
* 自动内存碎片整理，Ignite会尽可能的高效使用内存，然后在后台执行碎片整理以避免碎片化
* 改进性能和内存使用，不管是内存还是磁盘，所有数据和索引都以类似的页面化格式进行存储，避免不必要的数据序列化和反序列化

Ignite固化内存架构将内存拆分成固定大小的页面，这些页面会被存储于内存的堆外内存受管内存区（Java堆外），然后在磁盘上以特定的层次结构进行存储。Ignite在内存和磁盘上维护了同样的二进制格式，这样就不需要花费在内存和磁盘之间移动数据时进行序列化的开销，Ignite固化内存架构的架构图：



1. Data Regions

固化内存由一个或者多个堆外内存区组成，内存区通过内存策略配置的逻辑可扩展区域。这个区域大小可变、退出策略以及其他参数再下文描述。固化内存默认只会分配一个内存区，占用本集群节点可用内存的20%

1. Operational vs Historical Data

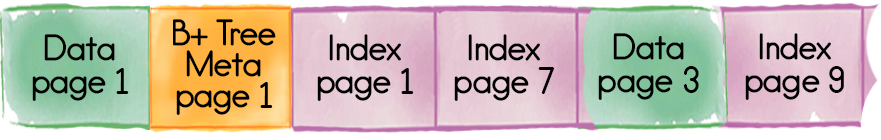
出于性能的考虑，需要尽可能将操作性数据保存在内存中，这时需要配置多个内存区，比如Person、Purchases和Records实体，分别存储于PersonsCache、PurchasesCache和RecordsCache缓存中。Person和Purchases的数据是操作性的，即这些数据会被频繁访问，而Records是访问量较少的历史数据。集群中使用200GB内存：

* Person和Purchases操作性或者频繁访问的数据，分配190GB的内存，在集群中会有更大的性能
* RecordCache，历史数据或者较少的数据分配10GB的内存，它的数据大部分会位于磁盘上

1. Memory Segment

每个内存区都开始于初始值，然后有一个可增长的最大值。这个区域扩展至其最大值的过程，都会被分配连续的内存段。内存区的最大值默认为系统可用物理内存的20%。

内存块是从操作系统获得的连续字节数组或者物理内存，这个块会被分为固定大小的页面，该块中可以驻留若干种不同类型的页面，如下图所示：



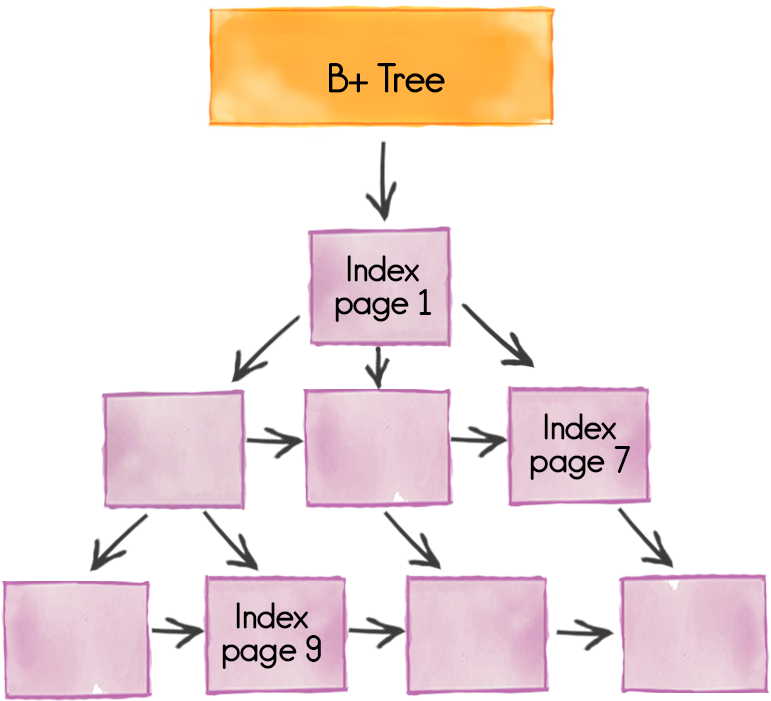
1. Data Pages

数据页面存储时从应用端插入Ignite缓存中的缓存条目（在上图中标注为绿色）。通常，一个数据页面持有多个键值条目，以更高效利用内存避免内存碎片化。当新键值条目加入缓存时，页面内存机制会查找适合该条目的页面然后加入里面。

当条目总大小达到MemoryConfiguration.setPageSize参数配置的页面大小时，该条目会占据多于一个数据页面。如果在更新期间条目的大小超过它所属的数据页面的可用空间，那么Ignite会搜索一个足够空间容纳更新后的条目的新数据页面，然后将数据移动到那里。

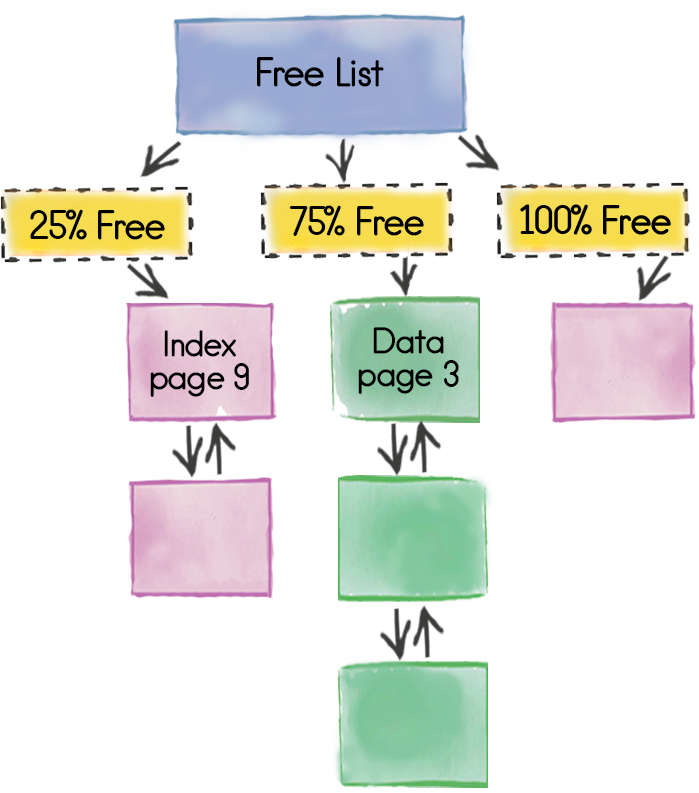
1. B+树和索引页面

应用定义和使用的SQL索引是以B+树数据结构的形式进行维护，对于一个SQL模式中声明的每个唯一索引，Ignite会实例化并且管理一个专用的B+树实例



1. 空闲列表

当调用myCache.put这种的操作时，Ignite会先从空闲列表中获取一个新条目，空闲类表是一个双向列表，存储了大致相当于空间空间的内存页面的引用。比如，有一个空闲列表，它存储了所有的数据页面，其占用最多75%的空闲空间，还有一个列表跟踪索引页面，它占用剩余的25%的容量，数据和索引页面是由独立的空闲列表来跟踪的



myCache.put(keyA，valueA)执行流程如下：

* Ignite找到myCache所属的内存区
* 在该内存区中，会定位持有myCache的键B+树的元页面
* 计算keyA的哈希值，然后在B+树中检索该键所属的索引页面
* 如果对应的索引页面在内存或者磁盘上都没有找到，那么会从空闲列表中申请一个新的页面成功之后，它就会被加入B+树
* 如果索引页面是空的（未指向任何数据页面），根据总的缓存条目大小会从空闲列表分配一个新的数据页面，然后在索引页面中添加到新数据页面的引用
* 该索引条目会加入该数据页面

1. 内存碎片整理

固化内存会将数据全部保存到数据页面的特定页面类型中，但是随着时间的推移每个独立的页面都可能由于键值条目的插入、更新和删除而更新多次，这就会导致整个内存碎片化，如下图：



页面有一个头部，保存所有的必要信息，所有的键值条目都是从右往左添加，保存页面中键值条目位置的偏移量是从左往右保存的，并且大小时固定的，偏移量作为指针用户定位页面

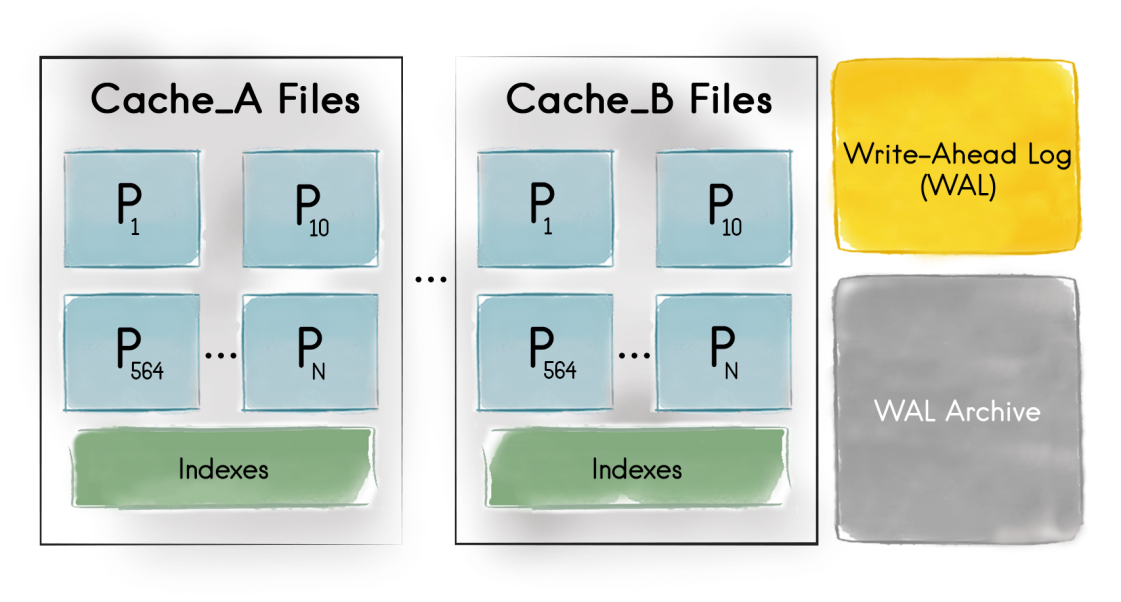
中的键值条目。中间的区域是空闲空间，当有新数据进入集群时就会被填充进入，如下图：



## **持久化**

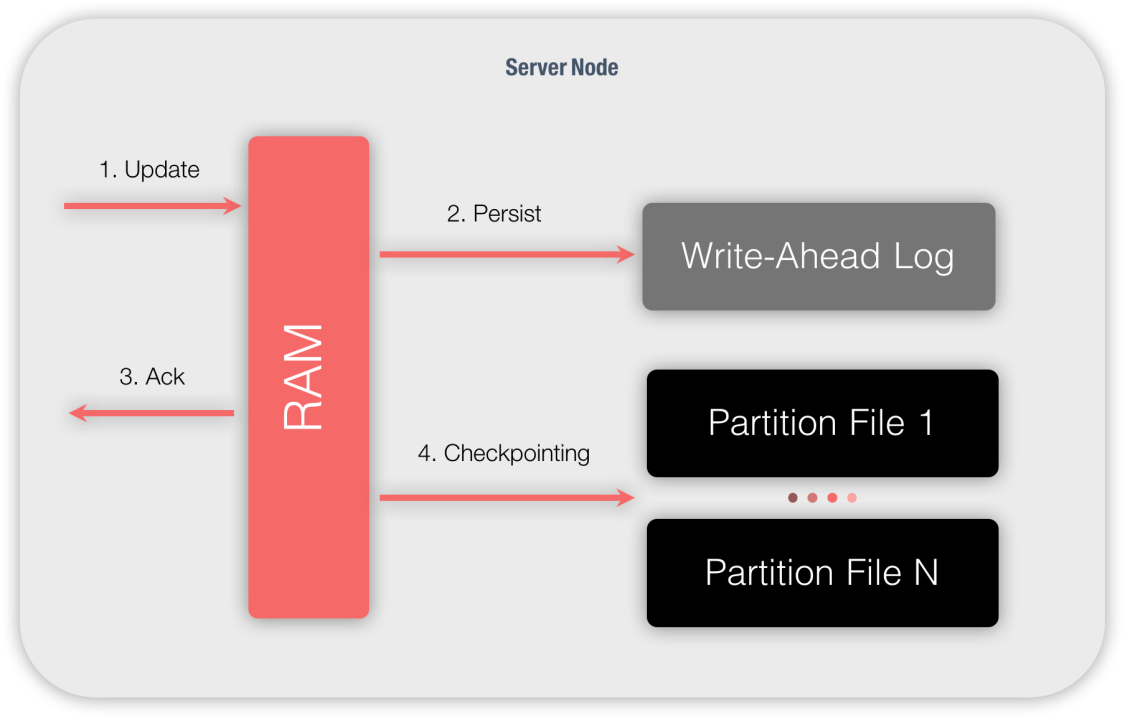
Ignite的原生持久化是一个分布式、支持ACID以及兼容SQL的磁盘存储，可以作为一个可选的磁盘层与Ignite的固化内存透明化的集成，然后将数据和索引存储在SSD、闪存、3D Xpoint以及其他类型的非易失性存储中。

打开Ignite的持久化之后，不需要将所有的数据和索引保存在内存中，或者在节点及集群重启后对数据进行预热，因为固化内存和持久化紧密耦合之后，将其视为一个二级存储层，当内存中数据和索引的一个子集如果丢失，固化内存会从磁盘上进行获取。当开启持久化存储后，内存和磁盘都会保存数据和索引，如下图所示：



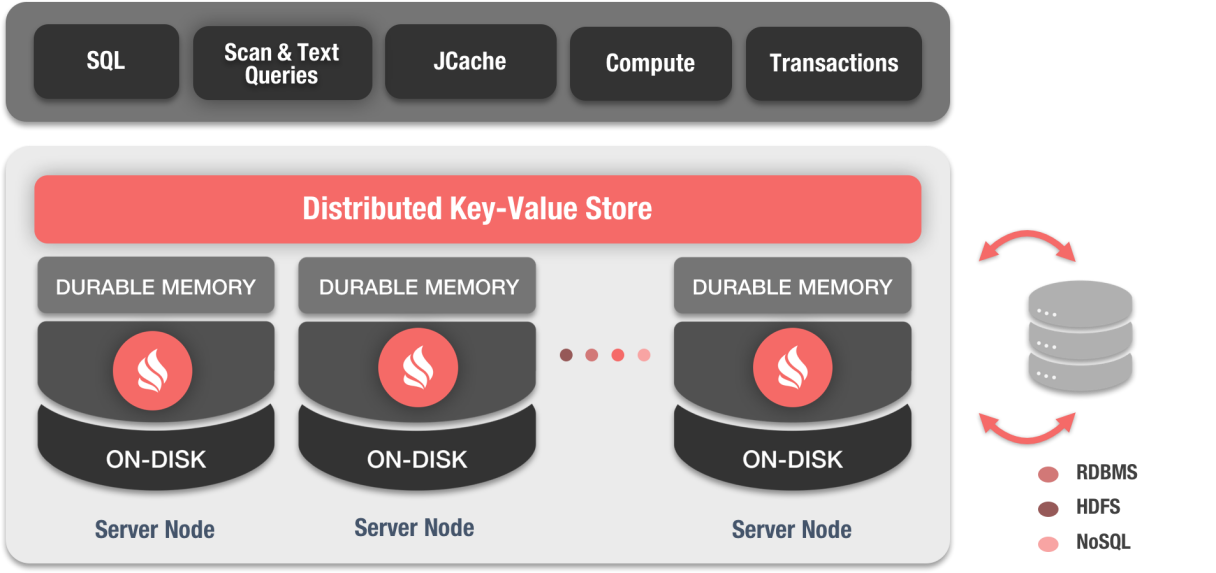
开启Ignite持久化以后，当更新Ram中的数据时，要先写入WAL文件，在Ignite中WAL文件顺序切分成多个文件Segment。

Ignite可以作为3rd party数据库（RDBMS）、Cassandra及MongoDB的缓存层，Ignite配置CacheLoader和CacheWriter来从这些数据库中读取数据，为了实现透明访问Ignite提供CacheStore



# **数据网格**

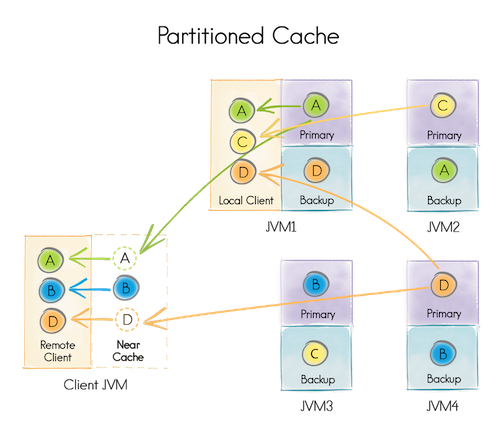
Ignite数据网格是基于内存的分布式键值存储，可以视为一个分布式的分区化哈希，每个集群节点都持有数据的一部分，增加集群节点就可以缓存更多的数据



与其他键值存储系统不同，Ignite通过可插拔的哈希算法来决定数据的位置，每个客户端可以通过一个加入哈希函数决定一个键属于哪个节点，而不需要任何特定的映射服务器或者Name节点。 Ignite数据网格支持本地、Replication和分区化的数据集，允许使用标准SQL语法方便地进行跨数据局查询，同时支持内存数据中进行分布式SQL关联。

* 1. **分区模式**

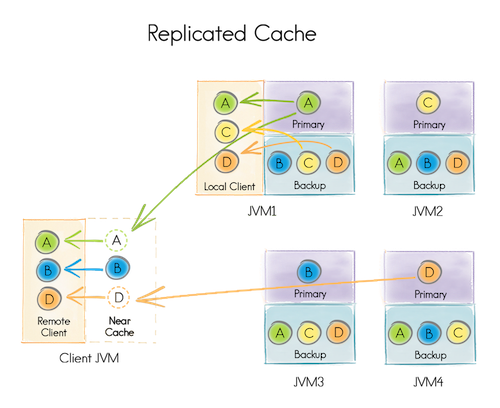
分区模式是扩展性最好的分布式缓存模式，在这种模式下，所有数据被均等分布在分区中，所有分区也被均等地拆分在相关的节点上，实际上就是为缓存的数据构建了一个巨大的内存分布式存储。这个方式可以在所有节点上只要匹配总可用内存就可以存储尽可能多的数据，但是读取时只有特定节点才持有缓存的数据，其适合于数据量很大而更新频繁的场合，下图是分区缓存的例子：



* 1. **复制模式**

复制模式中，所有数据都被复制到集群每个节点，这个缓存模式提供最大的数据可用性，然而这个模式每个数据更新都要传播到其他所有节点，因而会对性能和可扩展性产生影响。

Ignite中，复制缓存的实现类似于分区缓存，每个键都有一个主拷贝而且在集群内的其他节点也会有备份，如下图：



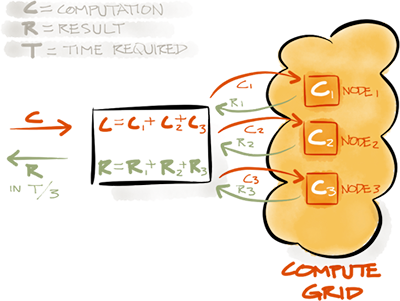
对于键A，运行于JVM1的节点为主节点，但是同时他还存储了其他数据的拷贝（B,C,D）。因为相同的数据被存储在所有集群节点中，复制缓存的大小受到节点的有效存储（内存和磁盘）的限制，这个模式适用于读缓存比写缓存频繁且数据集较小的场景，如果应用超过80%的时间用于查找缓存，那么就考虑使用复制缓存模式。

* 1. **本地模式**

本地模式是最轻量的模式，因为没有数据被分布化到其他节点，其适用于数据是只读或者定期刷新的场景中。当缓存数据失效需要从持久化存储中加载数据时，可以工作或者只读模式。除了分布化以外，本地缓存包括了分布式缓存的所有功能，比如自动数据回收、过期、磁盘交换、数据查询以及事务。

# **计算网格**

分布式计算是通过以并行的方式执行来获得高性能、低延迟和线性可扩展。Ignite计算网格提供了一套简单的API，使得可以在集群内的多台计算机上执行分布式计算和数据处理。分布式并行处理是基于在任何集群节点集合上进行计算和执行，然后将结果返回：



1）IgniteCompute

IgniteCompute接口提供了在集群节点或者一个集群组中运行很多中类型计算的方法，这些方法以一个分布式的形式执行任务或者闭包。只要至少一个节点有效，所有作业和闭包都会保证得到执行，如果一个作业的执行由于资源不足被提出，会提供一个故障转移机制。

*Ignite ignite = Ignition.ignite();*

*// Get compute instance over all nodes in the cluster.*

*IgniteCompute compute = ignite.compute();*

1. 分布式闭包

Ignite计算网格可以对集群或者集群组内的任何闭包进行广播和负载均衡，包括Java的runnables和callables。IgniteCompute使用broadcast方法将一个给定的作业广播到所有节点或则集群组中

*final Ignite ignite = Ignition.ignite();*

*IgniteCompute compute = ignite.compute(ignite.cluster().forRemotes()).withAsync();*

*compute.broadcast(() -> System.out.println("Hello Node: " + ignite.cluster().localNode().id()));*

*ComputeTaskFuture<?> fut = compute.future():*

*fut.listen(f -> System.out.println("Finished sending broadcast job."));*

1. Executor Service

IgniteCompute提供一个方便的API在集群中执行计算，Ignite提供了一个ExecutorService接口的分布式实现然后可以在集群中自动实现负载均衡的模式执行所有计算。该计算具有容错性以及保证只要有一个节点处于活动状态就能保证计算得到执行，可以将其视为一个分布式集群化线程池：

*ExecutorService exec = ignite.executorService();*

*// Iterate through all words in the sentence and create jobs.*

*for (final String word : "Print words using runnable".split(" ")) {*

*// Execute runnable on some node.*

*exec.submit(new IgniteRunnable() {*

*@Override public void run() {*

*System.out.println(">>> Printing '" + word + "' on this node from grid job.");*

*}*

*});*

1. MapReduce和ForkJoin

ComputeTask是Ignite对于简化内存内MapReduce的抽象，也非常接近于ForkJoin范式，其适用于处理离线的批量业务，然后对于内存内的数据进行计算时，实时性低延迟和高吞吐量通常具有更高的优先级。

ComputeTask定义了在集群内执行的作业以及这些作业到节点的映射，还定义了如何处理作业的返回值（Reduce），所有的IgniteCompute.execute方法都会在集群上执行给定的任务，应用只需要实现ComputeTask接口的map和reduce方法即可。

计算和数据的并置以最小化网络的数据序列化，以及可以显著提升应用的性能和可扩展性。不管何时，都应该尽力地将计算和缓存待处理数据的集群节点并置在一起。

# **服务网格**

服务网格可以在集群上部署任意用户定义的服务，比如自定义计数器、ID生成器、分层映射等，比如服务网格可以作为基于微服务架构的解决方案或者应用的技术基础。Ignite可以控制每个集群节点应该部署多少个服务的实例，可以自动的确保所有服务正确地部署和容错，如下图：



所有的服务网格功能都是通过IgniteServices接口实现的：

*Ignite ignite = Ignition.ignite();*

*// Get services instance spanning all nodes in the cluster.*

*IgniteServices svcs = ignite.services();*

*svcs.deploy(serviceCfg);*

# **消息和事件**

Ignite分布式消息可以在集群内所有节点间基于主题的通信，带有特定消息主题的消息可以分布到订阅了该主题的所有节点或者节点的子集。Ignite消息基于发布-订阅范式，发布者和订阅者通过一个通用的主题连接在一起，当一个节点针对主题T发布一个消息A，他会被分布到所有订阅主题T的节点。任意加入集群的新节点会自动订阅集群内其他节点订阅的所有准提。

Ignite分布式消息功能是通过IgniteMessaging接口提供的，其实例如下：

*Ignite ignite = Ignition.ignite();*

*// Messaging instance over this cluster.*

*IgniteMessaging msg = ignite.message();*

*// Messaging instance over given cluster group (in this case, remote nodes).*

*IgniteMessaging rmtMsg = ignite.message(ignite.cluster().forRemotes());*

1. 发布消息

send方法将一个带有特定消息主题的消息发送/发布到所有节点，消息可以有序或者无序的方式发送：

* sendOrdered，可以用于希望按照发送消息的顺序接收消息的场合，可以传递timeout参数来指定一个消息可以在队列中保存多长时间来等待发送之前的消息
* send，无序消息，不保证消息的顺序

1. 订阅消息

listen方法可以监听/订阅消息，当方法被调用时，带有指定消息主题的就会注册到所有节点来监听新的消息

* localListen，在本地注册一个带有指定主题的消息监听器然后监听来自集群内任意节点的消息
* remoteListen，在集群所有的节点上注册一个带有指定主题的监听器然后监听来自集群内任何节点的消息

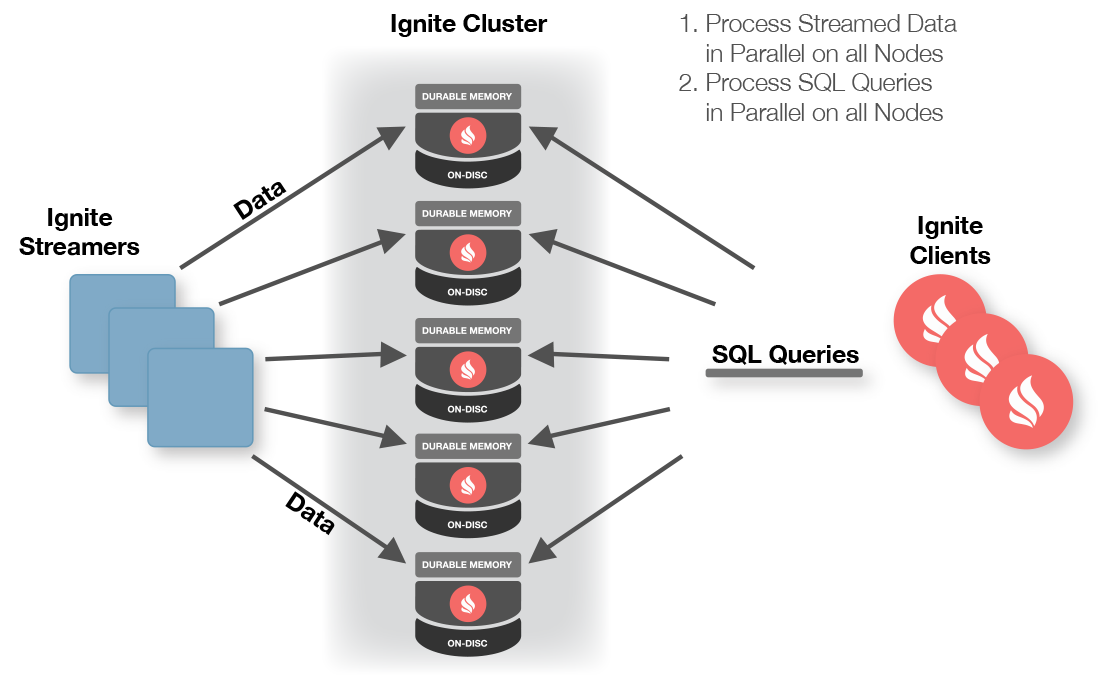
# **数据注入和流处理**

Ignite数据加载和流处理功能可以以可扩展及容错的方式处理持续不断的数据流或者在集群中预加载初始数据，在一个中等规模的集群中，数据注入Ignite或者预加载的速度很高，可以达到每秒处理百万级的事件

Ignite从持久化或者第三方存储这样的数据源进行数据的预加载，提供Ignite DataStreamer 和CacheStore API与主流技术集成。

* 数据流处理器，将大量连续数据注入Ignite缓存，以可扩展和容错的方式在数据被发送到集群节点之前通过把批量数据放在一起获取高性能
* IgniteCache，从磁盘或者第三方存储系统加载数据

其数据注入和处理如下图所示：

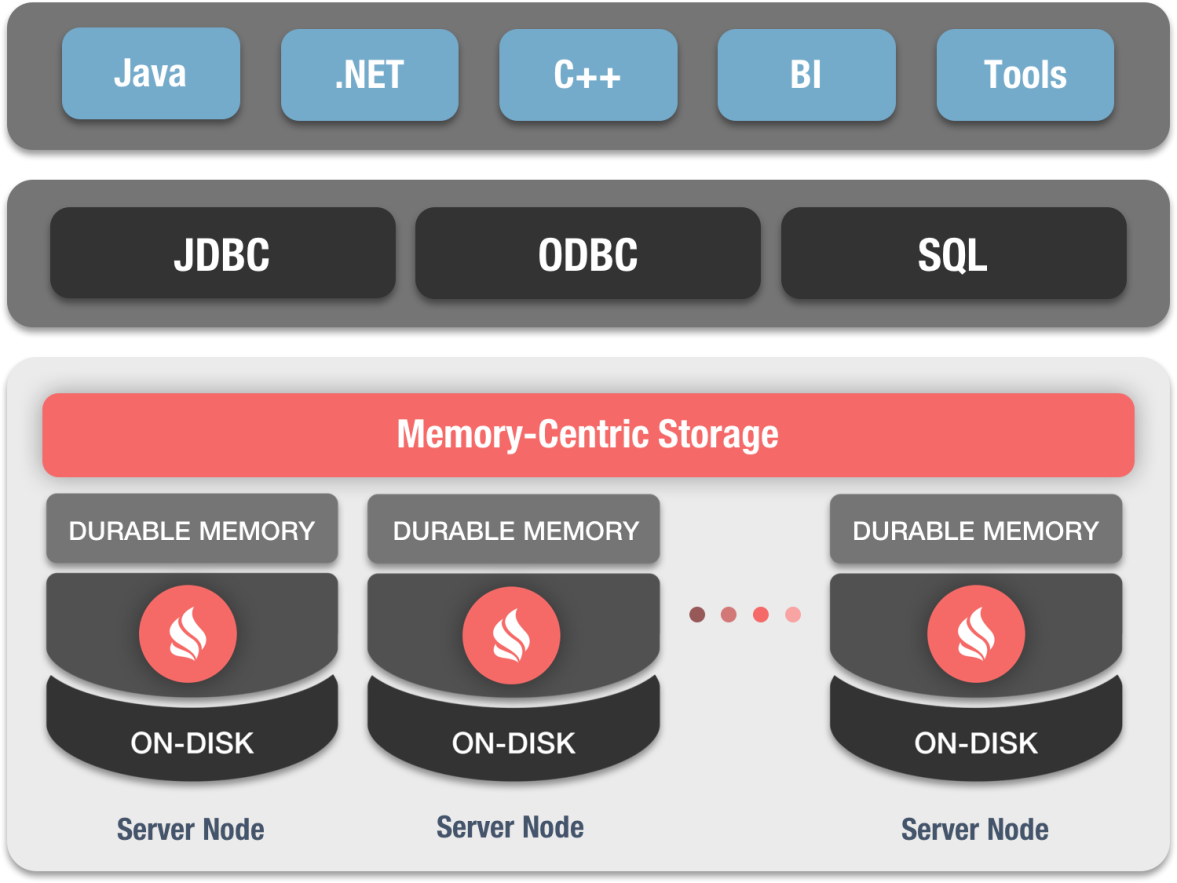


1. 客户端节点通过Ignite数据流处理器向Ignite注入有限或者持续的数据流
2. 数据在Ignite数据节点间自动分区，每个节点持有均等的数据量
3. 数据流可以在Ignite数据节点上以并置的方式直接并行处理
4. 客户端可以在数据流上执行并发SQL查询

Ignite可以与各种主要的流处理技术进行集成，比如Kafka、Camel、Storm及JMS，基于Ignite架构可以构建更强大的流处理功能。

# **SQL**

Apache Ignite是兼容ANSI-99、水平可扩展以及容错的分布式SQL数据库，这个分布式是以数据在集群范围的复制或则分区的形式提供的。Ignite支持所有的DML指令，包括SELECT、UPDATE、INSERT和DELETE，还实现一个与分布化系统有关的DDL指令子集。



Ignite完全支持分布式的SQL Join，包括并发和非并发的数据关联：并发关联在每个节点的可用数据集上执行，不需要在网络上移动大量的数据，这种方式为分布式数据库提供最好的扩展性和性能。可以通过外部的工具或者应用使用JDBC/ODBC驱动进行连接。

中文开发手册：https://www.zybuluo.com/liyuj/note/230739

内存架构：<https://www.zybuluo.com/liyuj/note/975720>

https://apacheignite.readme.io/docs/memory-architecture