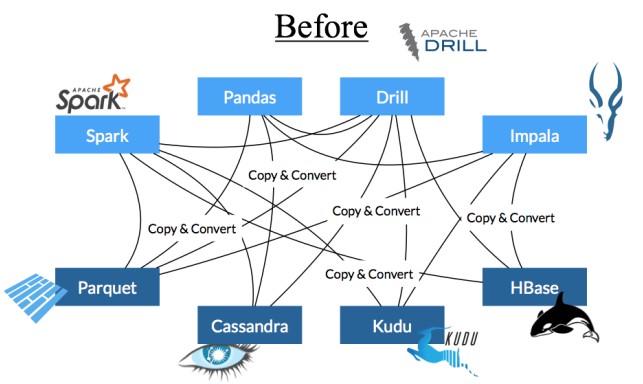
Apache Arrow

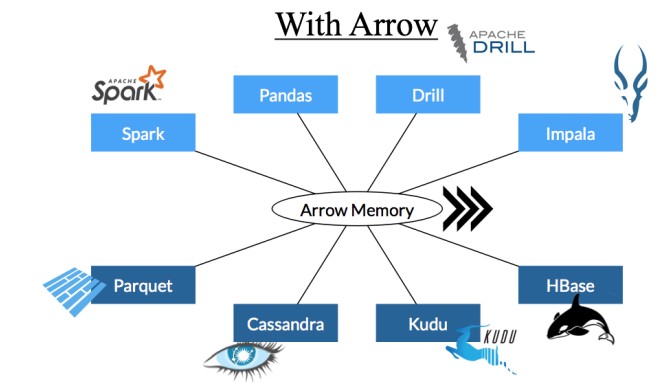
用户在使用Hadoop等大数据平台进行存储和数据处理时，比较关心系统的扩展性和性能，在这方面社区发布了很多的工具来完善数据分析，其涵盖各个层面，比如列式存储格式（Parquet/ORC）、内存计算层（Drill/Spark/Impala/Storm等）以及强大的API接口（Python和R语言）等。

Apache Arrow是一个跨平台的数据层，用于加快大数据分析项目的运行速度。提高大数据分析性能的一个重要手段是列式数据的设计和处理，列式数据的处理可以借助向量计算和SIMD充分挖掘硬件的潜力。Apache Drill数据查询引擎是将数据以列式方式存储在磁盘和内存容。Arrow是由Drill中Value Vector这一数据格式发展而来，其也支持关系型和动态数据集，并且提供良好的序列化和反序列化性能。

在众多分布式系统中，每个系统都有自己内部的内存格式，数据在不同系统的内存进行传递时，大部分的时间浪费在序列化和反序列化过程，这些功能没有一个标准，而且类似功能在多个项目中重复实现：



Apache Arrow提供了一种标准的数据交换格式，所有系统都使用相同的内存格式，实现不同系统之间数据的无缝共享，避免了系统间通信的开销，新的架构如下：



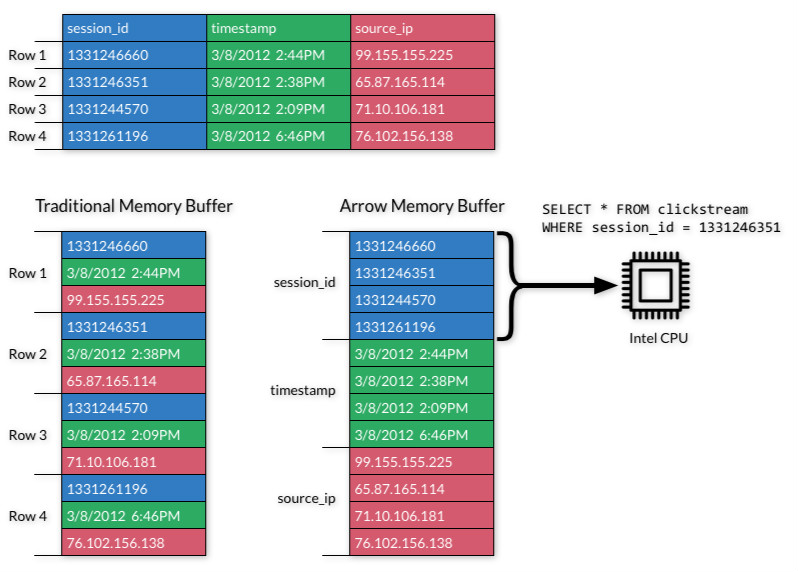
Arrow提供良好的扩展性，提供C++/Java版的实现，目前支持的项目包括Spark、Hadoop、R、Python/Pandas等。其具有以下的优势：

* 列式的内存布局可以使得随机访问的速度达到O(1)，在处理分析流和允许SIMD优化的现代处理器上非常高效
* 系统之间的数据交互非常高效，而且避免了数据的序列化和反序列化的消耗
* 使用类JSON的数据描述数据，可以支持复杂的数据类型

Arrow不是一个独立的系统，而是作为一个组件，在其他系统中嵌入该模块。

# Arrow数据存储格式

Arrow的基本思想是把向量在内存中的布局紧凑化，把按行表示的数据按列进行组织



**1）示例1**

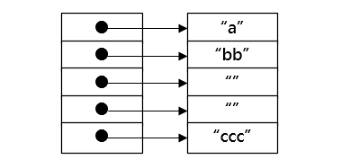
对于一个字符串向量：

*{“a”,”bb”,”,”,”,”,”ccc”}*

通常在C++中采用vector<string> 进行存储

*vector<string> strings = {"a", "bb", "", "", "ccc"};*

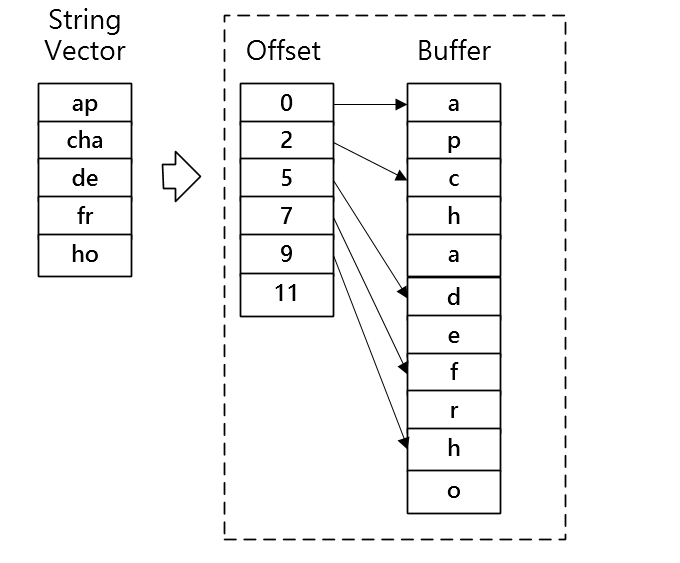
采用这种存储方式，其内存布局大致为：



在STL容器的String中存储字符串的空间在堆栈分配，那么每一个String对象会单独在对上申请内存空间，导致在内存中Vector<String>存储空间并非连续的，带来两个问题：

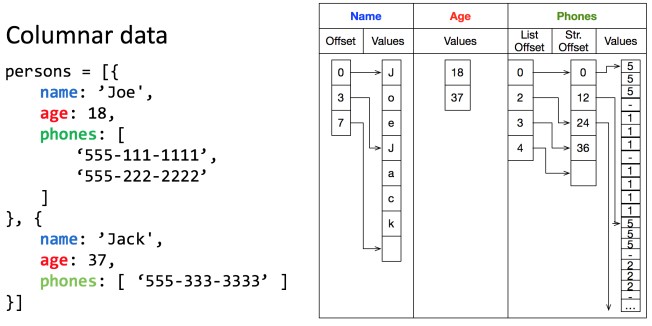
* 不能使用CPU Cache
* 序列化和反序列化代价较大

Apache Arrow将Vector进行变形，转换成如下的存储方式：



使用连续的内存空间Buffer时，引入了Offset向量对连续的Buffer空间进行索引，从而提供和vector<String>类似的取值接口。

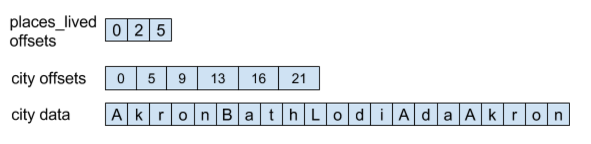
1. 示例2



3）示例3

|  |
| --- |
| *people = [*  *{*  *name: ‘mary’, age: 30,*  *places\_lived: [*  *{city: ‘Akron’, state: ‘OH’},*  *{city: ‘Bath’, state: OH’}*  *]}.....*  *]* |

在Arrow中，数组织people.places\_lived.city的访问看起来如下：



在Arrow中记录了placed\_live字段和city字段的偏移量，通过这个偏移量获得字段的值。

# 数据转换

在Apache Hadoop生态系统中，数据的存放方式：

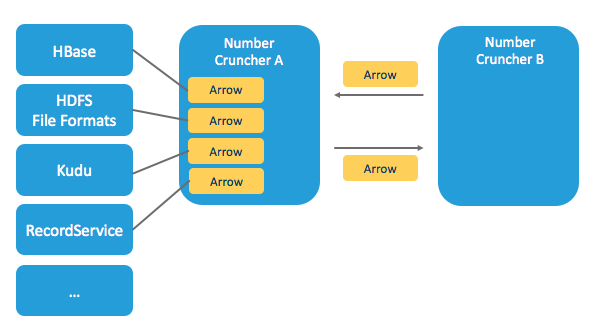
* 二进制格式：使用Apache Parquet、Text格式（Sequence file）或者CSV/TSV
* 结构化数据：Apache Cassandra、HBase、Kudu等存储系统

当使用Apache Impala/Spark等计算引擎从这些存储系统中请求数据到达系统后要先转换成本系统的格式后才能进行分析，一般来说在引擎使用的内存数据格式都是特定的，要使用Adapter将数据源数据（文件或者编排的数据）转换成本引擎处理的数据格式。

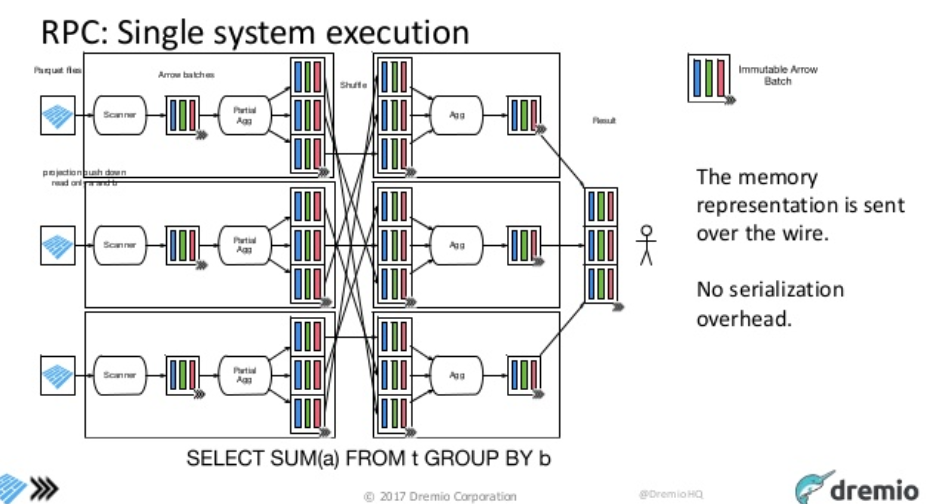
在集群中可以通过使用Arrow将数据移动流程改善为如下：

* 两个进程可以使用Arrow的数据描述形式来从另外一个进程获取数据，这个过程中不需要序列化和反序列化数据。例如Spark直接将Arrow数据发送到Python进程中来执行数据的处理
* 可以从Arrow-enabled的类数据库系统获取Arrow数据，例如Kudu将Arrow数据发送到Impala中

使用Arrow内存列数据格式的方式如下图所示：



下图是使用Parquet和Arrow之间数据转换的RPC交互的流程图



# Apache Arrow的使用

下面介绍Arrow的使用示例，将数据按照Arrow的格式写入数据并读取，源码见：

*<https://github.com/animeshtrivedi/ArrowExample>*

1. 定义ArrowSchema

*private Schema makeSchema() { //定义各列的名称，数据类型等信息*

*ImmutableList.Builder<Field> childrenBuilder = ImmutableList.builder();*

*childrenBuilder.add(*

*new Field("int",FieldType.nullable(new ArrowType.Int(32,true)),null));*

*childrenBuilder.add(*

*new Field("long", FieldType.nullable(new ArrowType.Int(64, true)), null));*

*childrenBuilder.add(*

*new Field("binary",FieldType.nullable(new ArrowType.Binary()),null));*

*childrenBuilder.add(*

*new Field("double",FieldType.nullable(new ArrowType.FloatingPoint(SINGLE)),*

*null) );*

*return new Schema(childrenBuilder.build(), null);*

*}*

1. ArrowWrite核心代码

*public void writeData() throws Exception {*

*this.batchSize = 100;*

*arrowFileWriter.start(); //ArrowFileWriter开始文件的写入*

*for(int i = 0 ; i < entries;) {*

*int toProccessItems = Math.min(this.batchSize, this.entries - 1);*

*root.setRowCount(toProccessItems);*

*for (Field field : root.getSchema().getFields()) {*

*//VectorSchemaRoot，定义的ArrowSchema*

*FieldVector vector = root.getVector(field.getName());*

*switch (vector.getMinorType()) {*

*case INT:*

*writeFieldInt(vector, i ,toProccessItems);*

*break;*

*case BIGINT:*

*writeFieldLong(vector, i , toProccessItems);*

*break;*

*case VARBINARY:*

*writeFieldVarBinary(vector, i , toProccessItems);*

*break;*

*case FLOAT4:*

*writeFieldFloat4(vector, i , toProccessItems);*

*break;*

*default:*

*throw new Exception("Not Supported yet type:" + vector.getMinorType());*

*}*

*}*

*}*

1. 读取数据ArrowReader

*public void makeRead(String filename) throws Exception {*

*File arrowFile = Utils.validateFile(filename, true);*

*FileInputStream fileInputStream = new FileInputStream(arrowFile);*

*DictionaryProvider.MapDictionaryProvider provider =*

*new DictionaryProvider.MapDictionaryProvider();*

*ArrowFileReader arrowFileReader =*

*new ArrowFileReader(new SeekableReadChannel(fileInputStream.getChannel()),*

*this.ra);*

*VectorSchemaRoot root = arrowFileReader.getVectorSchemaRoot();*

*List<ArrowBlock> arrowBlocks = arrowFileReader.getRecordBlocks();*

*for(int i = 0 ;i < arrowBlocks.size(); i++) {*

*ArrowBlock rb = arrowBlocks.get(i);*

*List<FieldVector> fieldVectors = root.getFieldVectors();*

*for (int j = 0 ;j < fieldVectors.size(); j++) {*

*Types.MinorType mt = fieldVectors.get(j).getMinorType();*

*switch (mt) {*

*case INT: showIntAccessor(fieldVectors.get(j)); break;*

*case BIGINT: showBigIntAccessor(fieldVectors.get(j)); break;*

*case VARBINARY: showVarBinaryAccessor(fieldVectors.get(j)); break;*

*case FLOAT4: showFloat4Accessor(fieldVectors.get(j)); break;*

*case FLOAT8: showFloat8Accessor(fieldVectors.get(j)); break;*

*default: throw new Exception(" MinorType " + mt);*

*}*

*}*

*}*

*}*

参考文献：

https://www.slideshare.net/julienledem/the-columnar-roadmap-apache-parquet-and-apache-arrow

https://www.dremio.com/webinars/columnar-roadmap-apache-parquet-and-arrow/

使用Arrow,Parquet构建关系型缓存：

https://www.dremio.com/webinars/apache-arrow-calcite-parquet-relational-cache/

https://www.iteblog.com/archives/1582.html

https://www.kdnuggets.com/2017/02/apache-arrow-parquet-columnar-data.html

https://github.com/apache/arrow/tree/master/java

http://arrow.apache.org/blog/

https://mapr.com/blog/apache-arrow-new-level-performance-and-interoperability-big-data-analytics/

http://blog.cloudera.com/blog/2016/02/introducing-apache-arrow-a-fast-interoperable-in-memory-columnar-data-structure-standard/