12

交互式处理

12 交互式处理

- Dremel数据模型与存储结构
- 并行查询
- Drill

12.1 Dremel数据模型与存储结构

- 数据模型

大数据交互式分析的计算架构主要包括三个方面：数据结构、存储体系、计算模型。数据结构是指计算模型采用的特殊设计的数据格式及组装方式，比如MapReduce采用键值对（key-value pair），Spark采用分布式弹性数据集（RDD），Dremel采用的是嵌套数据结构（nested data structure）。

Dremel采用了与XML[5]，JSON[6]这类数据描述语言相类似的一种数据格式Protocol Buffer[7]，它是Google的一个开源项目,用于结构化数据的序列化转换，不绑定于任何编程语言或平台，比XML更小、更快、也更简单，用户可基于Protocol Buffer定义自己的数据结构，然后使用自动生成的解码器程序来方便地读写这个数据结构。一个Protocol Buffer格式文件内容如下：

12.1 Dremel数据模型与存储结构

- 数据模型

```
message Document {
    required int64 DocId;
    optional group Links {
        repeated int64 Backward;
        repeated int64 Forward;
    }

    repeated group Name {
            repeated group Language {
            required string Code;
            optional string Country;
        }
        optional string Url;
    }
}
```

12.1 Dremel数据模型与存储结构

该段数据结构定义了如下内容：

一个Protocol Buffer格式的消息Document

该消息包含三个字段：1个int64字段、2个group类型字段

其中的repeated, required, optional是字段限制符，共有三类:

required: 必须赋值的字段

optional: 可有可无的字段

repeated: 可重复字段(变长度)

Protocol Buffer数据格式可用数学公式表达为：

π = dom | <A_1: π[*|?], ..., An: π[*|?]>

这里，π是一个数据类型，而Protocol Buffer文件可包含一个或多个数据类型。

π有两种可能（“|”是OR的意思）：一种是基本类型dom（如int，float，string等）；另一种是使用递归方式定义的，即π可以由其他定义好的π组成，A_1...An是这些π变量的命名。

“*”表示π包含的变量可以是重复型（repeated）即有多个，“？”表示是可选型（optional），即不包含任何元素。

12.1 Dremel数据模型与存储结构

在Protocol Buffer中可定义如下的嵌套数据类型：

```
message SearchResponse
{
    message Result
    {
        required string url = 1;
        optional string title = 2;
        repeated string snippets = 3;
    }
    repeated Result result = 1;
}
```

其中，Result是嵌套在SearchResponse中的一个数据结构。如果在SearchResponse之外另有一个变量要使用Result，可采用parent-name.child-name的形式调用，如下所示：

```
{
  optional SearchResponse.Result result = 1;
}
```

12.1 Dremel数据模型与存储结构

● 存储结构

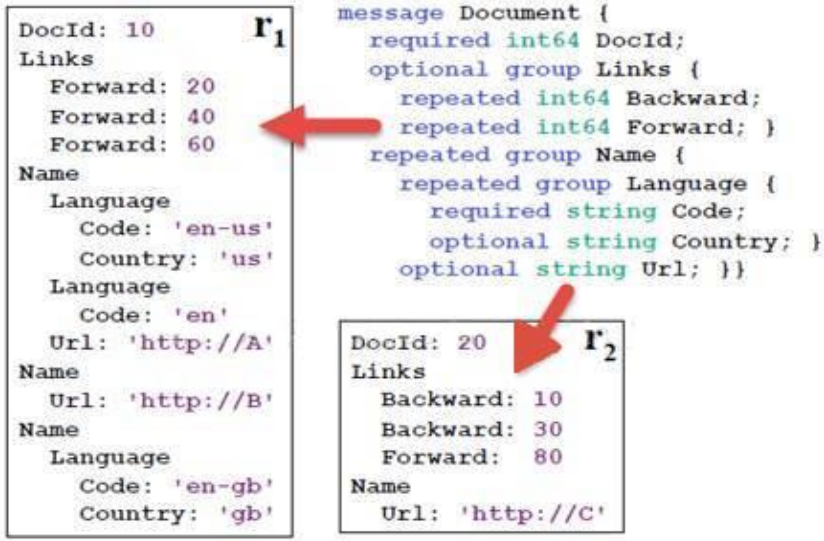
在讨论存储结构之前，我们先定义如下的概念：

数据记录（record）：指一条完整的嵌套数据，如果是数据库中，一条记录就是一行（row）数据。

值域或字码段（field）：值域或字码段在大部分情况下指的是同一个概念，是嵌套数据结构中的一个子项或元素，在数据表中就是一个列。

列（column）：数据结构中的一个值域或字码段在存储时就是一个列。

对于前述的Document嵌套数据类型，右图的代码可以产生如下两条数据记录（即Protocol Buffer文件包含的消息数据），即图中的r1和r2。



12.1 Dremel数据模型与存储结构

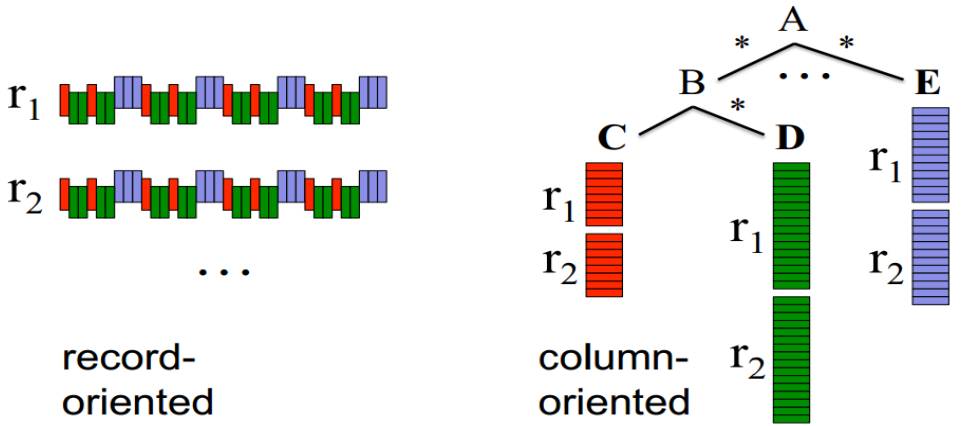
上述数据记录实际上可以用表格形式表示为：

DocId	Links				Name			
	Backword		Forward		Language		Url	...
					Code	Country		...
10	null	...	20	...	en-us	us	http://A	...
20	10	...	80	...	null	null	http://C	...
...	...		...	...	...	...	...	...

上述二维表数据在空间存储时有两种方式：行存储（row-oriented storage）和列存储（column-oriented storage）。行存储是以数据表的行键（RowKey）为基准、以数据记录（record）为单位进行存储，每一行数据包含了一个对象或事务的完整记录，每一行记录包含了多个值域（图右边r1和r2的不同颜色块表示不同的值域）。列存储则是将不同记录的相同值域（r1和r2的相同颜色块）放入一个列中存储，采用的是树状存储结构。

12.1 Dremel数据模型与存储结构

如果是行存储，在读取数据时（查找一条记录的某个值域）需要完成两个步骤：i）纵向按行键（RowKey）查找到该行；ii）横向向右搜索，跳过不相关值域，直至找到查询项。这种存储方式使得每读一个RowKey后，都需要跳到下一个RowKey的位置，所有要搜索的字段都不是连续存放，且有些值域是变长度的字符串（repeated）,不能通过简单公式计算得到地址，查询起来效率非常低。而如果按列存储方式，只需按树状结构找到需要查询列（column）第一个值域的首地址，然后顺序读取数据（每个record对应值域的地址飘移值（offset）都记录在元数据表中），不需要扫描其他不相干的列，不仅实现简单，而且磁盘顺序读取比随机读取要快得多，而且更容易进行优化（比如把临近地址的数据预读到内存，对连续同类型数据进行压缩存放），效率大大提高。



12.1 Dremel数据模型与存储结构

记录项r1和r2基于值域（列）被拆分成字码段，每一个字码段都用一个表存储，字码段名称保持了嵌套结构。r1和r2的嵌套数据结构包含DocId，Forward, Backword, Code, Country, Url等值域，按嵌套结构可以表示为：

DocId
Links.Forward
Links.Backward
Name.Language.Code
Name.Language.Country
Name.Url

DocId

value	r	d
10	0	0
20	0	0

Name.Url

value	r	d
http://A	0	2
http://B	1	2
NULL	1	1
http://C	0	2

Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2
80	0	2

Links.Backward

value	r	d
NULL	0	1
10	0	2
30	1	2

Name.Language.Code

value	r	d
en-us	0	2
en	2	2
NULL	1	1
en-gb	1	2
NULL	0	1

Name.Language.Country

value	r	d
us	0	3
NULL	2	2
NULL	1	1
gb	1	3
NULL	0	1

12.1 Dremel数据模型与存储结构

Dremel在将列存储树状结构映射到一维顺序存储时，需要考虑将来恢复嵌套数据结构如何满足下面两个要求：

列存储格式记录的无损表达(lossless representation of record structure in a columnar format)

嵌套数据结构的高速组装，即从列存储表恢复原有嵌套数据结构

Dremel采用了下面的Repetition Level和Definition Level定义及阅读器（reader）的有限状态机（FSM）设计来实现上述两个功能。

12.1 Dremel数据模型与存储结构

Repetition Level和Definition Level

Dremel采用的是列存储结构。对于图中的Document格式的数据记录r1，以其一个值域“Code”为例，其存储路径为：Name→Language→Code，其中Name和Language均是repeated类型，如图所示。

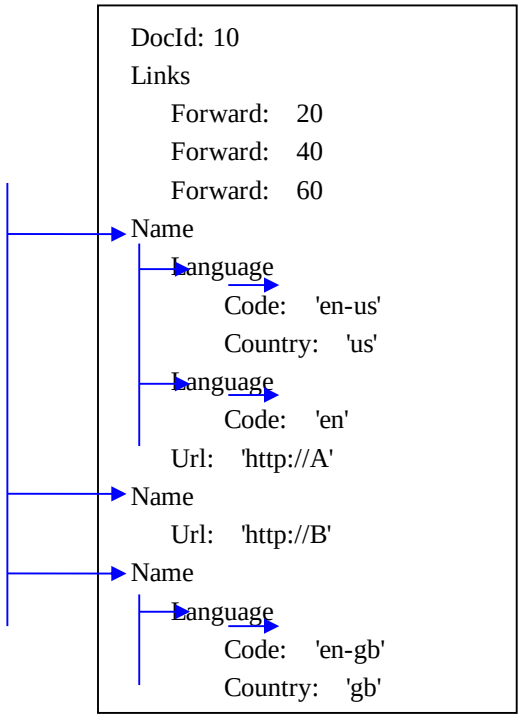
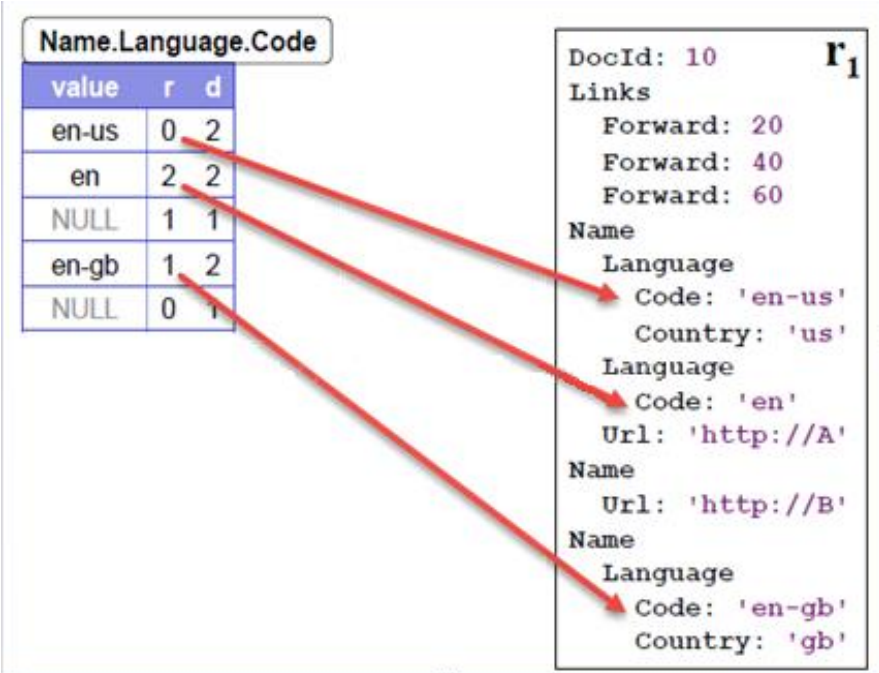


图 14-4　r1 的嵌套数据结构

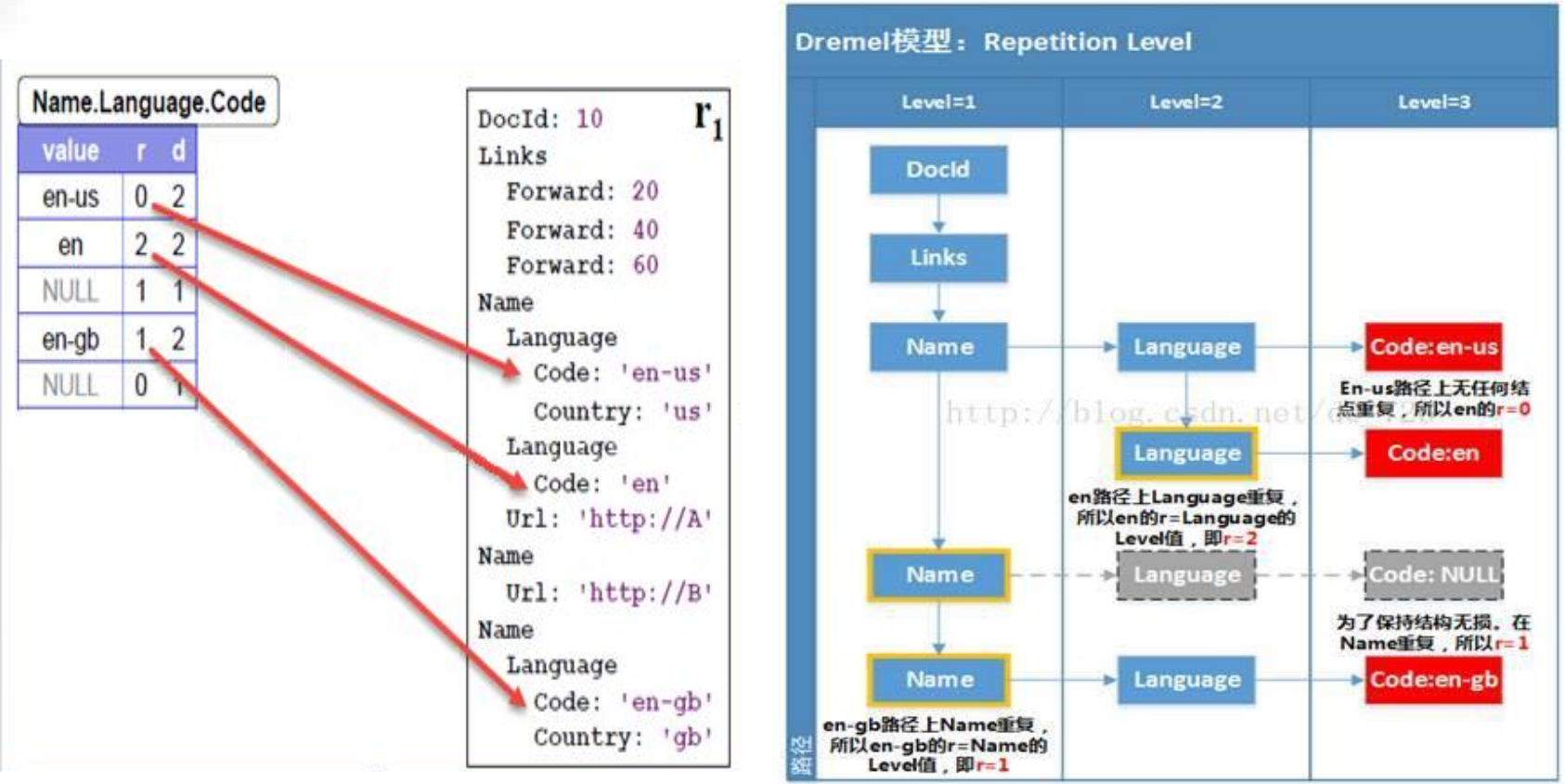
12.1 Dremel数据模型与存储结构

由于Dremel是列存储结构，因此Code在物理存储时单独作为一个列表存储，如图所示。

如果嵌套结构的字码段DocId，Name，Language，Code 可定义为不同的等级，则**Repetition Level可定义为：嵌套结构的一个最终值域的repetition level等于从最高等级字码段抵达此值域的路径上重复的字码段的等级；如果没有重复，则repetition level = 0。**



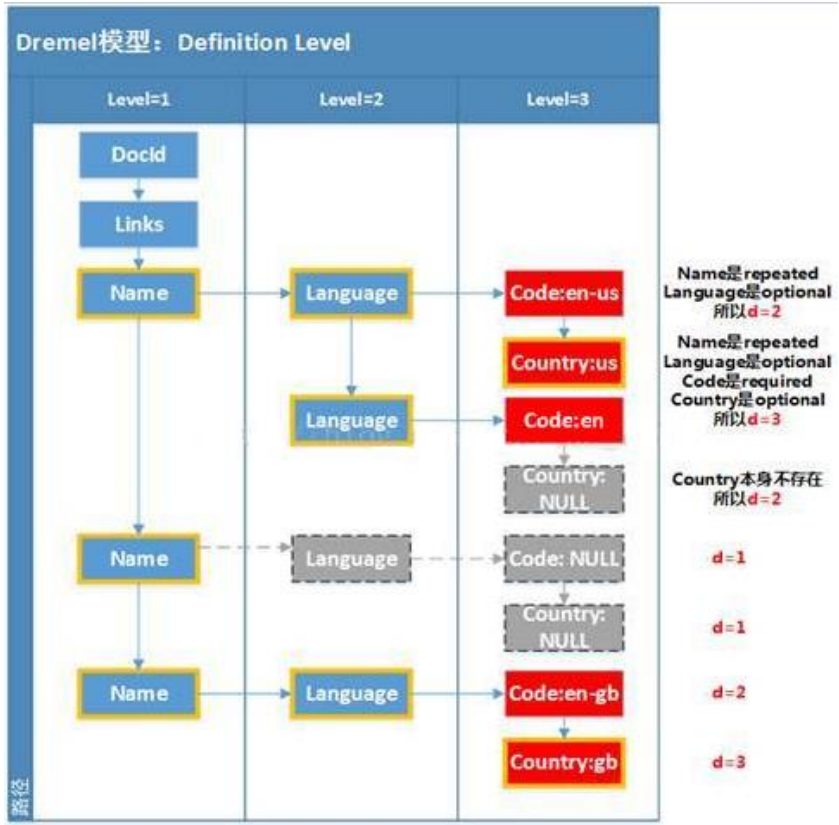
12.1 Dremel数据模型与存储结构



12.1 Dremel数据模型与存储结构

Definition Level

某一值域p的**Definition Level定义为：在抵达值域p的路径上，可能不存在类型（如optional型和repeated型）字码段却实际存在的数目。**



12.1 Dremel数据模型与存储结构

数据重构方法

基于上述repetition level和definition level的定义，Dremel可以方便地构建writer树并将嵌套数据结构拆分成多个列存储表进行存储。对于从顺序存储结构（物理存储）中重构处嵌套数据结构（逻辑结构），Dremel采用了如下的阅读器（reader）有限状态机（FSM，finite state machine）设计，以完成存储表到数据结构的快速重建。

在图所示的数据结构重建过程中，Dremel按照数据结构schema采用多个不同的阅读器（field reader）来读取并处理不同的字码段，对于每一个字码段FSM都从开始到结束循环一次。r值（repetition level）用于控制reader的转换（对不同的字码段使用不同的reader）。

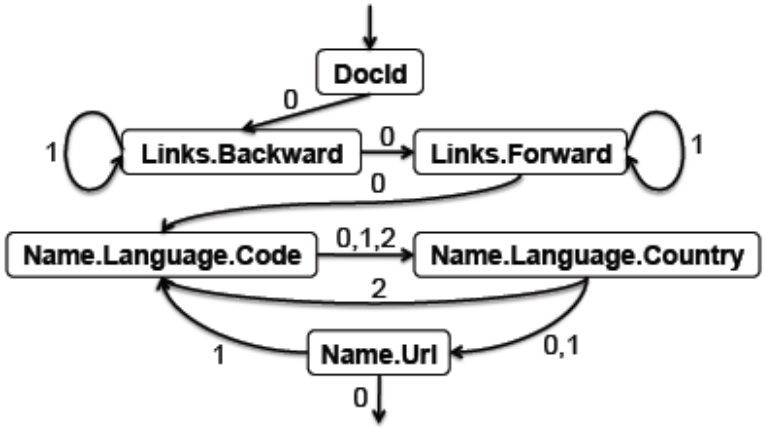


图 14-8　阅读器（reader）的 FSM

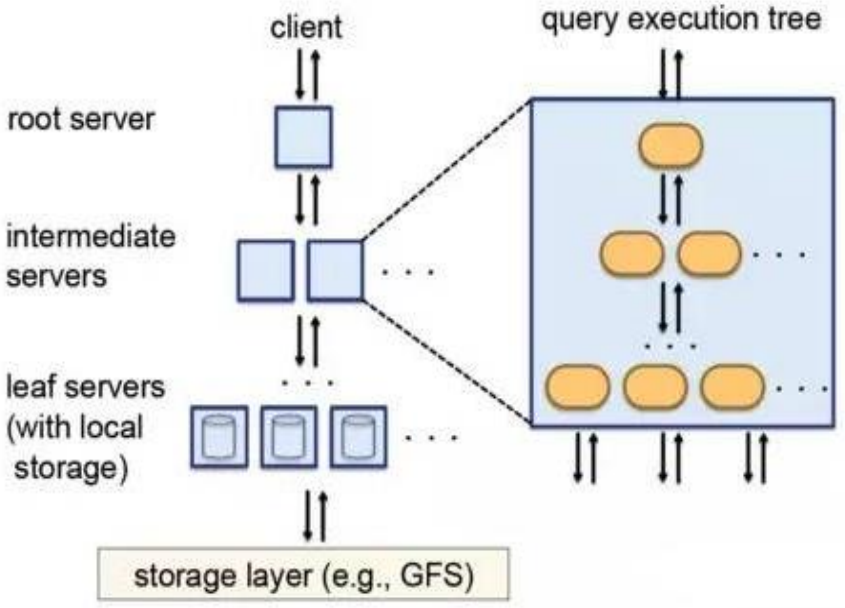
12.1 Dremel数据模型与存储结构

总结上述内容，Dremel的数据模型和存储结构的要点如下：

- Dremel采用了平台无关的数据格式Protocol Buffer来描述嵌套数据结构，这种嵌套数据结构提供了一种海量数据规模下的高效存储和读取方式；
- Dremel采用了基于值域的列存储结构，即将数据记录基于列拆分成多个列存储表，多个记录的相同值域的值存放在同一列存储表中。在物理存储时将多个列存储表进行顺序存储；
- Dremel的列存储表中不光包含各记录的列值，还包含对应的r值（repetition level）和d值（definition level）,Dremel对每个值域按照有限状态机（FSM）规则读取顺序存储的列存储表并进行数据记录的重构；
- 每次对顺序存储的物理表进行扫描和数据记录重建时，Dremel并不需要扫描和重建全部数据，而可根据需要只扫描部分数据、重建感兴趣的值域（列）。

12.2 并行查询

Dremel采用的是多层服务树（serving-tree）计算架构，如图所示。 Dremel集群最上层的根服务器（root server）接收所有的客户端查询请求，并把查询语句分解，读取相关元数据，再把分解后的请求下发中间服务器（intermediate server）。中间服务器进一步把查询需求分发到它所属的下级叶节点服务器（leaf server）完成并行计算。数据记录存储在叶节点服务器的本地文件系统上，叶节点完成计算处理后，其返回计算结果的过程与上述步骤逆向而行。



12.2 并行查询

服务树的计算构架与MapReduce的计算构架（Map/Shuffle/Reduce）相比，更适合于超大规模数据查询的筛选和聚合运算，执行速度更快，有如下两点原因：

- 列存储结构使得查询仅需扫描它关心的列存储表（字码段），而勿需扫描全部数据集
- 由于服务树架构，根节点和中间节点只起任务分解和结果汇聚作用，最后的计算处理是在叶节点进行，叶节点相互之间没有依赖关系，因此可以实现高并发度的并行处理

Dremel主要用于支持数据查询业务（并不擅长数据增删操作），这种列存储结构和服务树并行处理模式对查询操作性能的优化尤其明显。大于98%的查询操作响应时间低于10秒，响应时延超过10秒不到2%。这其中，某些查询任务扫描的数据记录数达到1000亿条。

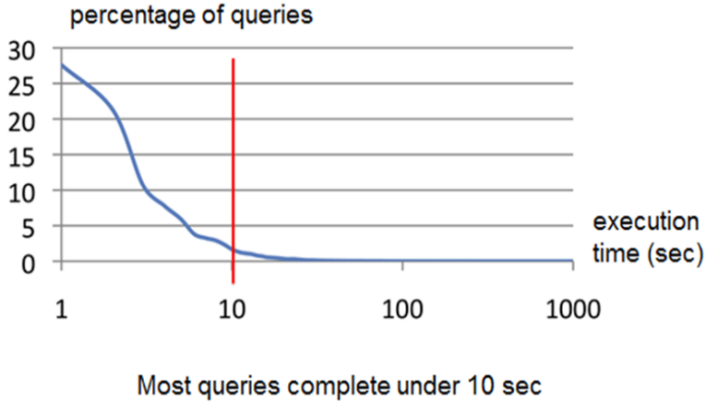
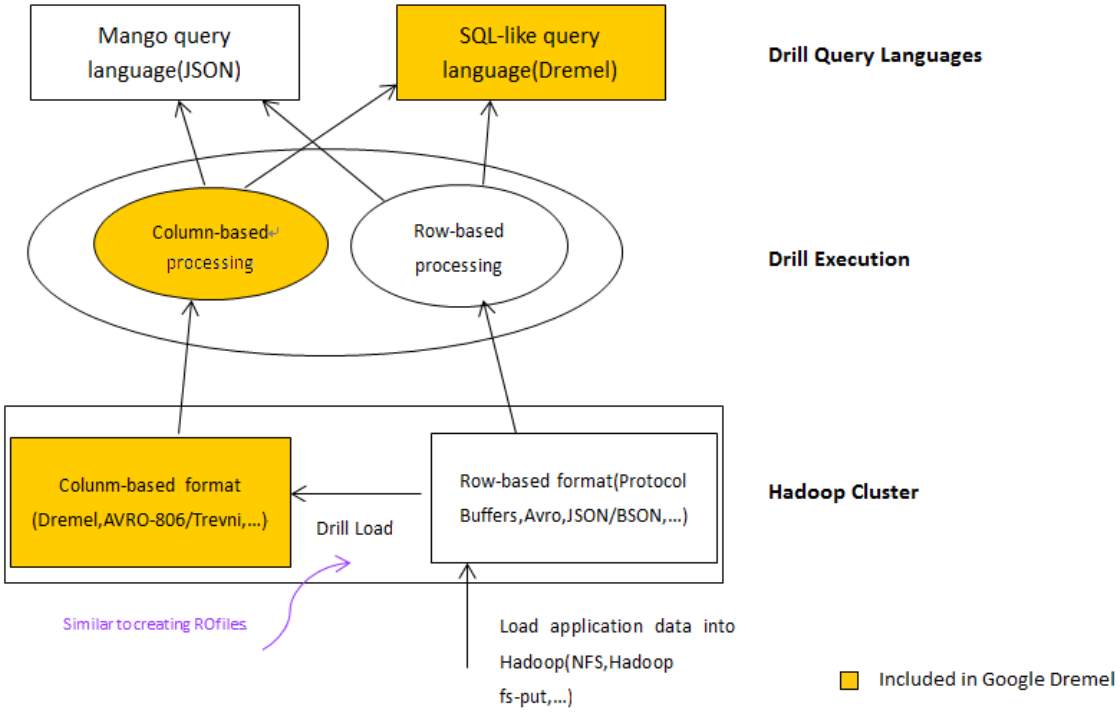


图 14-12 Dremel 的查询响应时间

12.3 Drill

Apache Drill的计算架构分为支持DrQL查询的客户端、Drill执行引擎、底层存储系统（Hadoop集群）三个层次，如图所示。在计算节点上Drill使用Hadoop/HDFS作为底层的数据存储系统。



12.3 Drill

Drill的软件架构如图所示，其核心是DrillBit服务单元，它负责接收客户端请求，处理查询，并将结果返回给客户端。DrillBit单元能够安装和运行在Hadoop集群各个节点上，形成一个分布式计算环境。DrillBit在节点运行时能够最大限度实现数据的本地化，不需要节点间的数据移动。Drill使用Zookeeper来进行集群节点管理和运行状态监控。尽管Drill多数情况下运行在Hadoop集群上，但它也可以运行在其他分布式集群上。

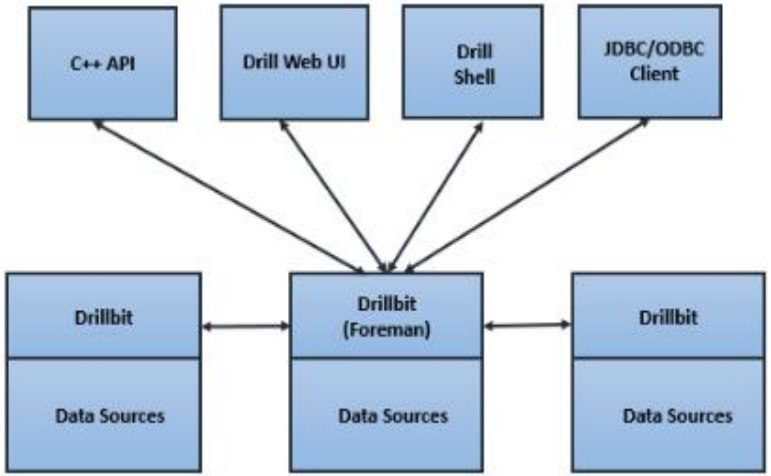


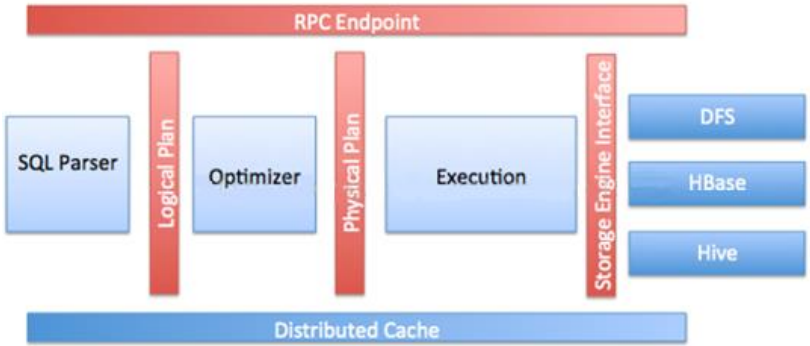
图14-15　Drill软件架构

12.3 Drill

DrillBit单元

Drill主要的软件单元DrillBit包含如下组件：

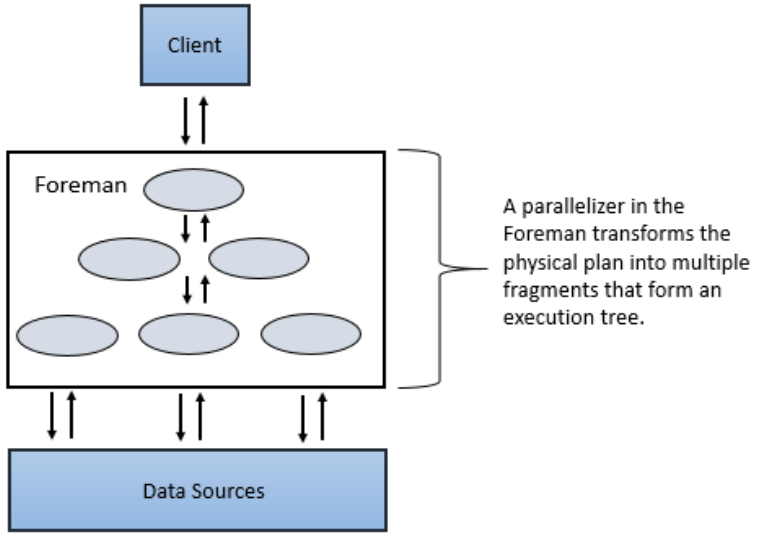
- RPC Endpoint: 一个提供低开销的基于Protobuf的RPC通信的组件。此外，Drill也提供C++编程接口和JDBC/ODBC连接界面用于用户程序与DrillBit的交互。
- SQL Parser：一个使用Optiq开源框架的SQL解析器，将SQL语句解析映射到对应的Drill objects，该解析器的输出是语言无关的。
- Optimizer：Drill执行语句优化器。
- Storage Plugin Interface：Drill作为多个数据源之上的查询层，它需要与底层不同类型的存储系统（分布式文件系统，HBase，Hive等）对接。



12.3 Drill

计算模型

与Dremel类似，Drill计算模型也采用了查询树结构，称为execution-tree,如图所示。客户端程序将查询请求（SQL query）提交给Foreman（树根节点root上的DrillBit），由Foreman启动整个查询过程的解析、优化、分发、计算执行流程，并负责将查询结构返回给客户端。



12.3 Drill

综合上述，可看出Drill在计算架构设计方面有如下特点：

◆ 动态模式检测（dynamic schema discovery）

Drill在启动查询过程时不需要预先声明数据类型和模式，Drill在执行过程中可以动态检测模式。

◆ 数据模型灵活（flexible data model）

Drill允许访问嵌套数据的字码段（列）,并提供直观的易扩展的操作。从数据模型的角度来看,Drill提供了一个灵活的分层列存储数据模型,可以处理复杂的、动态变化的数据类型。

◆ 分散元数据（de-centralized metadata）

Drill没有集中元数据的需求，因此不需要在一个元数据库来管理数据表和视图。Drill数据来源于存储插件对数据源的读取，而存储插件可以支持完整元数据(Hive)、部分元数据(HBase)、或没有集中元数据(文件系统)。没有集中的元数据意味着Drill可以同时读取和处理多种数据源。

◆ 可扩展性（extensibility）

Drill在所有层面都提供了可扩展的架构，包括存储插件、查询器、优化器、执行引擎和客户端API，用户可以自定义各个层面的组件来进行扩展。

End of Session
Thank you!