厦门大学计算机科学系研究生课程

《大数据技术基础》

第13章 Google Dremel

（2013年新版）

林子雨

厦门大学计算机科学系

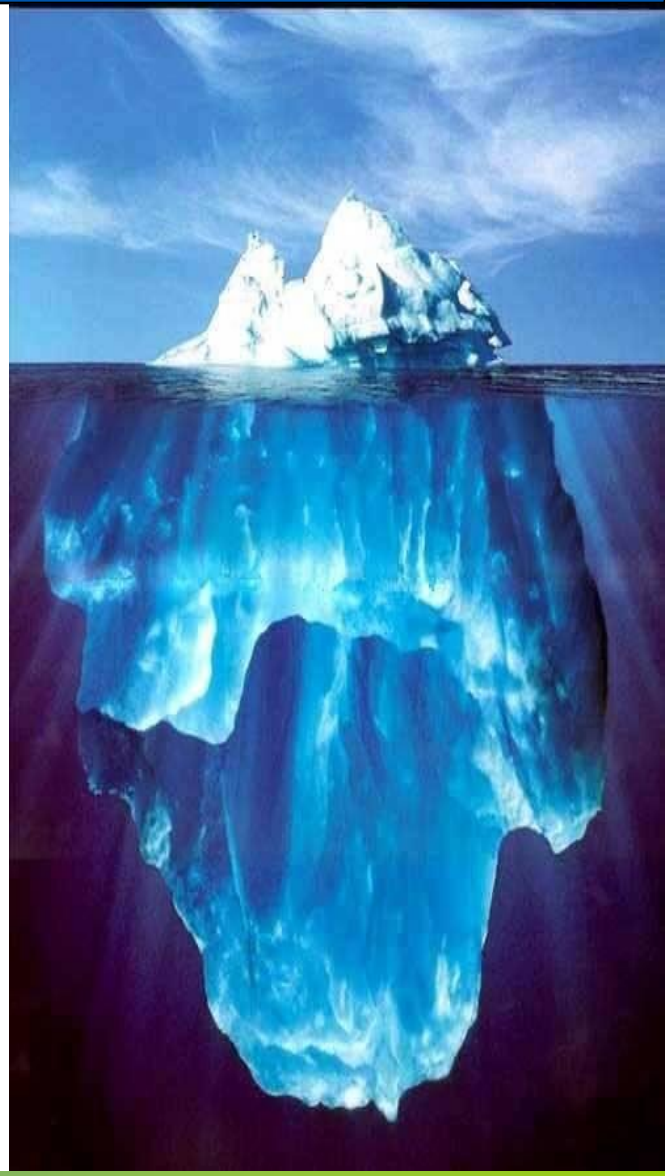
E-mail: ziyulin@xmu.edu.cn

主页：http://www.cs.xmu.edu.cn/linziyu

提纲

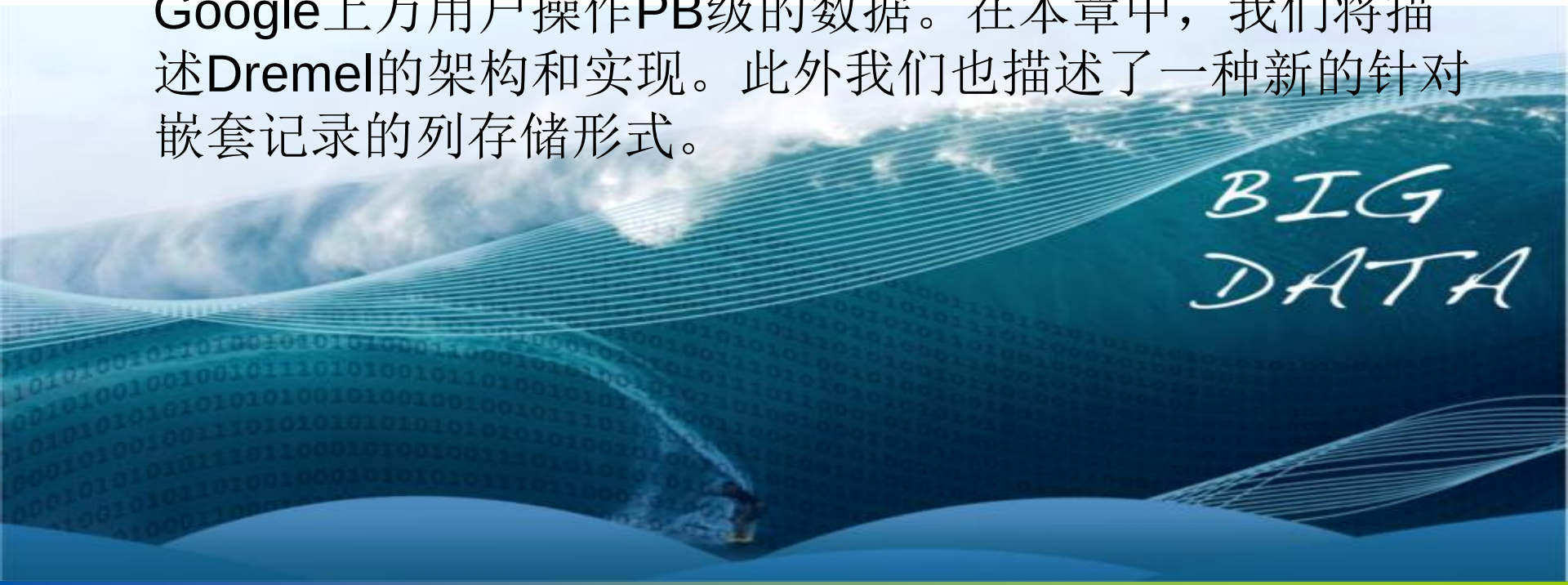
- Dremel概述：
  - 大规模数据分析
  - Dremel的特点
- Dremel的数据模型
- 嵌套列状存储：
  - 重复深度和定义深度
  - 分割记录为列状存储
  - 记录装配
- 查询语言
- QUERY的执行
- 小结

本讲义PPT存在配套教材，由林子雨通过大量阅读、收集、整理各种资料后编写而成
下载配套教材请访问《大数据技术基础》2013班级网站：http://dblab.xmu.edu.cn/node/423



Dremel 概述

Dremel是一种可扩展的、交互式的即时查询系统，用于只读嵌套（nested）数据的分析。通过结合多级树状执行过程和列状数据结构，它能做到几秒内完成对万亿张表的聚合查询。系统可以扩展到成千上万的CPU上，满足Google上万用户操作PB级的数据。在本章中，我们将描述Dremel的架构和实现。此外我们也描述了一种新的针对嵌套记录的列存储形式。



大规模数据分析

- 大规模分析型数据处理在互联网公司乃至整个行业中都已经越来越广泛。不仅仅是因为目前已经可以用廉价的存储来收集和保存海量的核心业务数据。

- 执行大规模交互式数据分析对并行计算能力要求很高。

- 互联网和科学计算中的数据经常是没有关联的。因此，在这些领域一个灵活的数据模型是十分必要的。规格化、重新组合这些互联网规模的数据通常是十分耗时的。嵌套数据模型成为了Google处理大部分结构化数据基础。

Dremel的特点

（1） Dremel是一个大规模、稳定的系统。在一个PB级别的数据集上面，将任务缩短到秒级，无疑需要大量的并发。Google一向是用廉价机器办大事的好手。但是机器越多，出问题概率越大，如此大的集群规模，需要有足够的容错考虑，保证整个分析的速度不被集群中的个别慢(坏)节点影响。

（2） Dremel是MR交互式查询能力不足的补充。和MapReduce一样，Dremel也需要和数据运行在一起，将计算移动到数据上面。在设计之初，Dremel并非是MapReduce的替代品，它只是可以执行非常快的分析，在使用的时候，常常用它来处理MapReduce的结果集或者用来建立分析原型。

Dremel的特点

（3） Dremel的数据模型是嵌套(nested)的。互联网数据常常是非关系型的。Dremel还需要有一个灵活的数据模型，这个数据模型至关重要。而传统的关系模型，由于不可避免的有大量的Join操作，在处理如此大规模的数据的时候，往往是有心无力的。

（4） Dremel中的数据是用列式存储的。使用列式存储，分析的时候，可以只扫描需要的那部分数据的时候，减少CPU和磁盘的访问量。同时列式存储是压缩友好的，使用压缩，可以综合CPU和磁盘，发挥最大的效能。

（5） Dremel结合了Web搜索 和并行DBMS的技术。首先，他借鉴了Web搜索中的“查询树”的概念，将一个相对巨大复杂的查询，分割成较小较简单的查询。其次，和并行DBMS类似，Dremel可以提供了一个SQL-like的接口，就像Hive和Pig那样。

Dremel数据模型

π = dom | <A1 : π[*|?],…,An : π[*|?]>

π是字段数据的类型，可以是原子类型或记录类型。

Ai 表示字段标记

Required、optional（？）
Repeated（*）表示字段类型

```
DocId: 10                    r1
Links
  Forward: 20
  Forward: 40
  Forward: 60
Name
  Language
    Code: 'en-us'
    Country: 'us'
  Language
    Code: 'en'
  Url: 'http://A'
Name
  Url: 'http://B'
Name
  Language
    Code: 'en-gb'
    Country: 'gb'
```

```
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; }}
```

```
DocId: 20                    r2
Links
  Backward: 10
  Backward: 30
  Forward:  80
Name
  Url: 'http://C'
```

Figure 2: Two sample nested records and their schema

Dremel数据模型

嵌套数据模型为Google的序列化、结构化数据奠定了一个平台无关的可扩展机制。代码生成工具生成具体编程语言（如C++、Java）的代码。跨语言的兼容性通过对记录的标准二进制化表示来保证，记录中的字段及相应的值序列化后传输。通过这种方式，Java写的MR程序可以处理另外一个C++生成的数据源中的记录。因此，如果记录采用列存储，快速的重建对于MR和其它数据处理工具都非常重要。

嵌套列状存储

如图1所示，我们目标是连续的存储一个给定的字段的所有值来改善检索效率。本节将介绍：

- 一个列状格式记录的无损表示
- 分割记录
- 记录装配

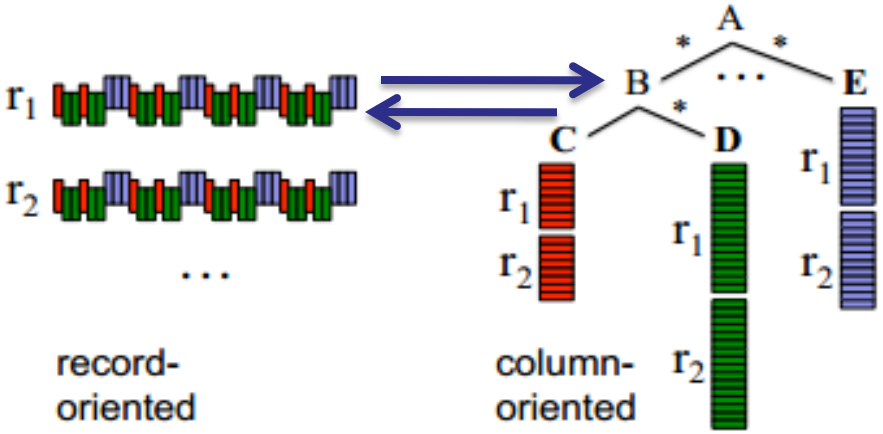


Figure 1: Record-wise vs. columnar representation of nested data

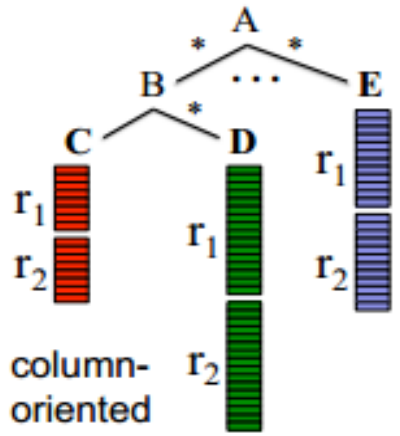
重复深度、定义深度

重新审视一下图1右边的列状存储结构，这是Dremel的目标，它就是要将图2中Document那种嵌套结构转变为列状存储结构。

```
DocId: 10                    r1
Links
  Forward: 20
  Forward: 40
  Forward: 60
Name
  Language
    Code: 'en-us'
    Country: 'us'
  Language
    Code: 'en'
  Url: 'http://A'
Name
  Url: 'http://B'
Name
  Language
    Code: 'en-gb'
    Country: 'gb'
```

```
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; }}
```

```
DocId: 20                    r2
Links
  Backward: 10
  Backward: 30
  Forward: 80
Name
  Url: 'http://C'
```



lumnar representation of nested data

Figure 2: Two sample nested records and their schema

重复深度、定义深度

重复深度是记录该列的值是在哪一个级别上重复的。举个例子说明：对于Name.Language.Code 我们一共有三条非Null的记录。

➢第一个是”en-us”，出现在第一个Name的第一个Lanuage的第一个Code里面。在此之前，这三个元素是没有重复过的，都是第一个。所以其R为0。

➢ 第二个是”en”，出现在下一个Lanuage里面。也就是说Lanague是重复的元素。Name.Language.Code中Lanague排第二个，所以其R为2.

➢第三个是”en-gb”，出现在下一个Name中，Name是重复元素，排第一个，所以其R为1。

要注意第二个Name在r1中没有包含任何Code值。为了确定‘en-gb’出现在第三个Name而不是第二个，我们添加一个NULL值在‘en’和‘en-gb’之间。

r_1

```
DocId: 10
Links
  Forward: 20
  Forward: 40
  Forward: 60
Name
  Language
    Code: 'en-us'
    Country: 'us'
  Language
    Code: 'en'
  Url: 'http://A'
Name
  Url: 'http://B'
Name
  Language
    Code: 'en-gb'
    Country: 'gb'
```

r_2

```
DocId: 20
Links
  Backward: 10
  Backward: 30
  Forward:  80
Name
  Url: 'http://C'
```

重复深度、定义深度

为什么引入定义深度？

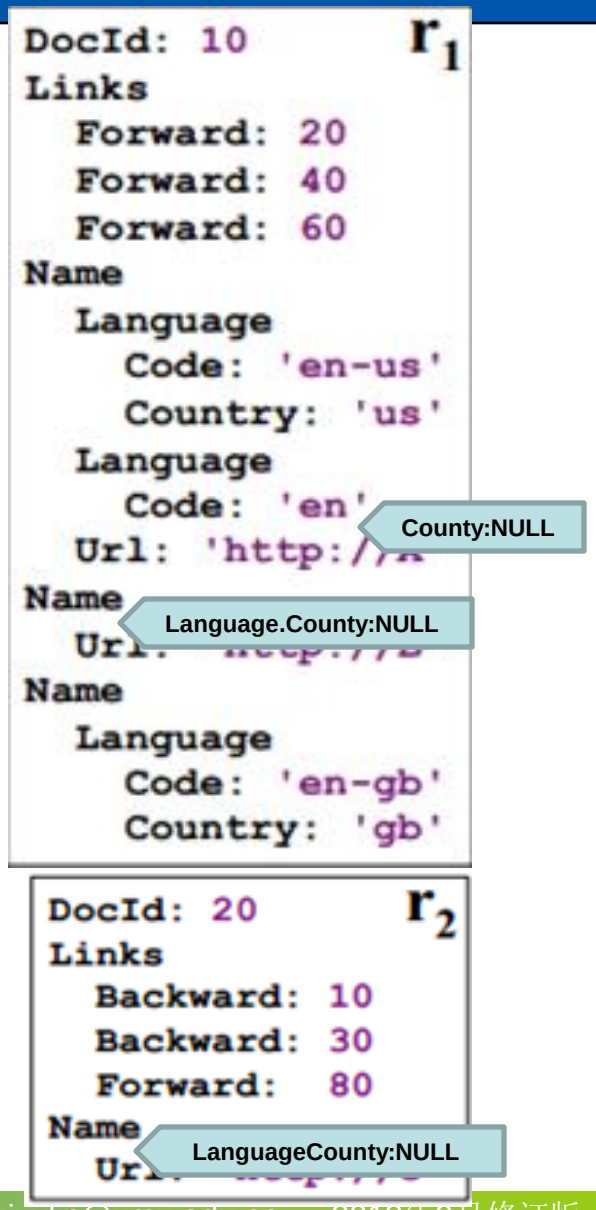
所有列都是先存储r1，后存储r2，也就是说对所有的列，记录存储的顺序是一致的。知道它们是否属于同一条记录，这也是保证记录被拆分之后不会失真的一个重要手段。既然顺序是十分必要的不能失真的因素，那当某条记录的某一列的值为空时就不能简单的跳过，必须显式的为其存储一个NULL值，以保证记录顺序有效。而NULL值本身能诠释的信息不够，比如记录中某个Name.Language.Country列为空，那可能表示Country没有值，也可能表示Language没有值，这两种情况在装配算法中是需要区分处理的，不能失真，所以才需要引出定义深度，能够准确描述出此信息。

重复深度、定义深度

定义深度，用来记录该列是否是”想象”出来的。所以对于非NULL的记录，是没有意义的，其值为为非required字段的个数。例如Name.Language.Country,

- “us”在r1里面，其中Name,Language,Country是有定义的。所以D为3。
- 第一个“NULL”也是在r1的里面，其中Name,Language是有定义的，其他是想象的。所以D为2。
- 第二个“NULL”还是在r1的里面，其中Name是有定义的，其他是想象的。所以D为1。
- “gb”是在R1里面，其中Name,Language,Country是有定义的。所以D为3。
- 第三个“NULL”在r2中，Name是有定义的，其他事想象的，所以D为1

如果路径中有required，可以将其减去，因为required字段必然会被赋值，记录其数量没有意义。



重复深度、定义深度

```
DocId: 10                    r1
Links
  Forward: 20
  Forward: 40
  Forward: 60
Name
  Language
    Code: 'en-us'
    Country: 'us'
  Language
    Code: 'en'
  Url: 'http://A'
Name
  Url: 'http://B'
Name
  Language
    Code: 'en-gb'
    Country: 'gb'
```

```
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; }}
```

```
DocId: 20                    r2
Links
  Backward: 10
  Backward: 30
  Forward: 80
Name
  Url: 'http://C'
```

Figure 2: Two sample nested records and their schema

注意：在计算重复深度的范围时，
最大值为重复字段的个数。

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

Figure 3: Column-striped representation of the sample data in Figure 2, showing repetition levels (r) and definition levels (d)

重复深度、定义深度

几点细节：

（1）每列存储为一组块。每个块包括重复深度和定义深度（以及压缩的字段值。

（2）NULL是由定义级别来决定的，所以它不会显式地存储。字段路径对应的定义级别小于路径上可选和可重复字段个数总和的即可认为是NULL。

（3）如果值总是被定义的，那么它的定义级别也不会被存储。类似的，重复深度只在必要时存储。比如，定义深度 0 意味着重复深度 0，所以后者可省略。

分割记录

```
DocId: 10                    r1
Links
   Forward: 20
   Forward: 40
   Forward: 60
Name 0
   Language 0
      Code: 'en-us'  0
      Country: 'us'  0
   Language 2
      Code: 'en' 2
   Url: 'http://A'   0
Name 1
   Url: 'http://B'   1
Name
   Language
      Code: 'en-gb'
      Country: 'gb'
```

seenField：

Name

Language

Code

Country

Code

Url

Url

记录装配

给定到一个字段的子集，我们的目标是重组原始记录就好像他们只包含选择的字段，其他字段就当不存在。核心想法是：我们为字段子集创建一个有限状态机（FSM），读取字段值、重复深度、定义深度，然后顺序地将值添加到输出结果上。一个字段的FSM状态对应这个字段的reader。状态的变化标记上了重复深度。一旦一个reader获取了一个值，我们将查看下一个值的重复深度来决定状态如何变化、跳转到哪个reader。对于每一条记录，FSM都是从开始状态到结束状态变化一次。

记录装配

三个重点：

- 第一，所有数据都是按图3那种类似一张张“表”的形式存储的；
- 第二，算法会结合schema，按照一定次序一张张地读取某些“表”（不是所有的，比如只统计Forward那就只会读取这一张“表”），次序是不固定的，这个次序也就是状态机内状态变迁的过程；
- 第三，无论次序多么不固定，它都是按记录的顺序不断循环的（比如当前数据按顺序存储着r1,r2,r3… 那会进入第一个循环读取并装配出r1，第二个循环装配出r2…），一个循环就是一个状态机从开始到结束的生命周期。

记录装配

```
DocId: 10                    r1
Links
  Forward: 20
  Forward: 40
  Forward: 60
Name
  Language
    Code: 'en-us'
    Country: 'us'
  Language
    Code: 'en'
  Url: 'http://A'
Name
  Url: 'http://B'
Name
  Language
    Code: 'en-gb'
    Country: 'gb'
```

```
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; }}
```

```
DocId: 20                    r2
Links
  Backward: 10
  Backward: 30
  Forward: 80
Name
  Url: 'http://C'
```

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

Figure 2: Two sample

triped representation of the sample data in Fig-
tition levels (r) and definition levels (d)

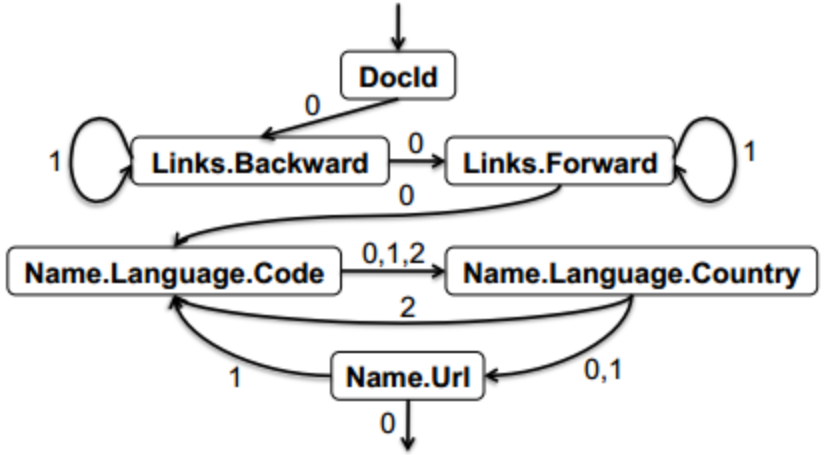
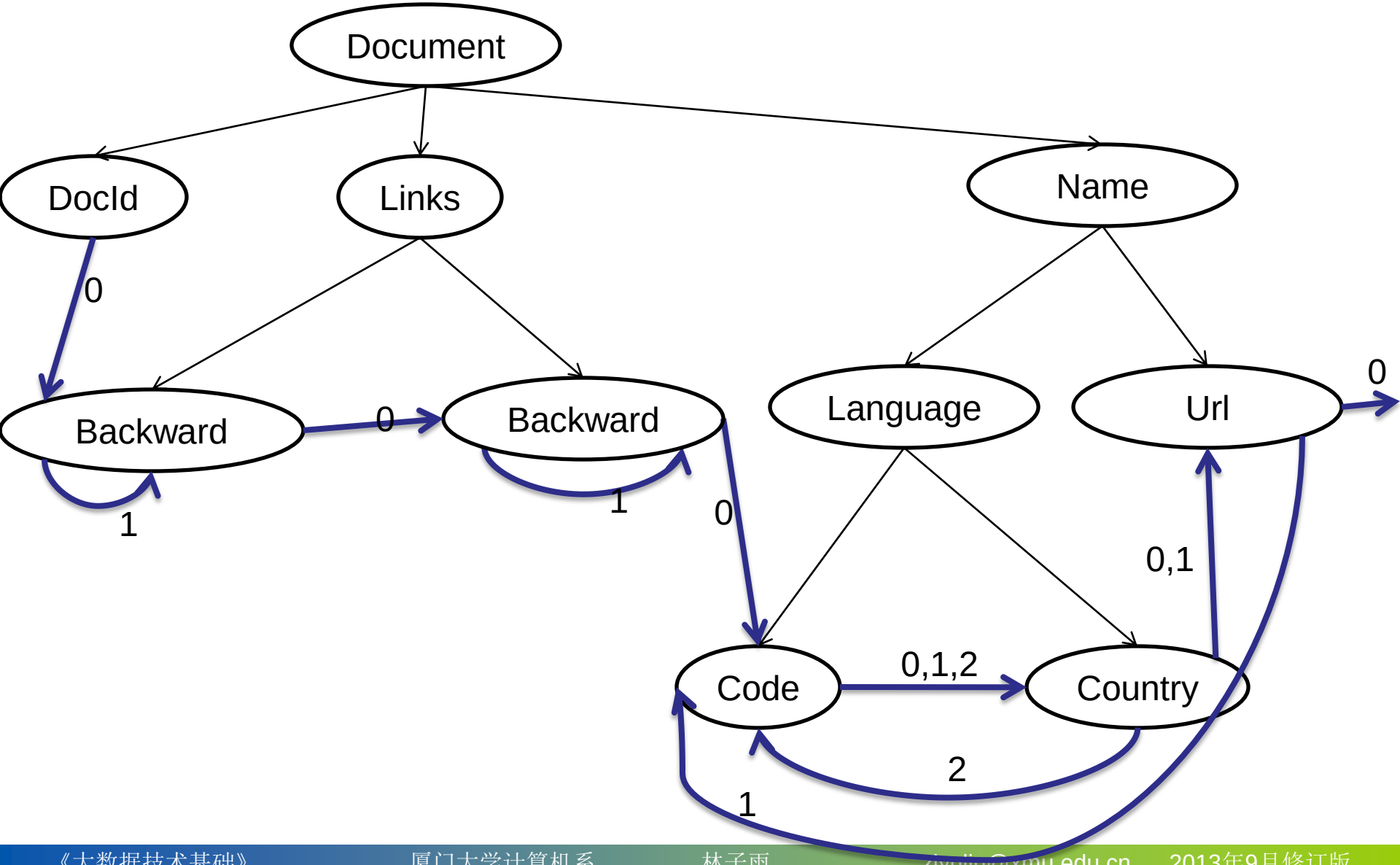


Figure 4: Complete record assembly automaton. Edges are labeled with repetition levels.

FSM的构造


Document
DocId
Links
Name
0
Backward
0
Backward
Language
Url
0
1
1
0
0,1
0,1,2
Code
Country
2
1

查询语言

Dremel的查询语言基于SQL，可在列状嵌套存储上高效执行。每个SQL语句（被翻译成代数运算）以一个或多个嵌套表格和它们的schema作为输入，输出一个嵌套表格和它的schema。图6描述了一个查询例子，执行了投影、选择和记录内聚合等操作。例子中的查询执行在图2中的t = {r1,r2}表格上。字段是通过路径表达式来引用。

```
SELECT DocId AS Id,
  COUNT(Name.Language.Code) WITHIN Name AS Cnt,
  Name.Url + ',' + Name.Language.Code AS Str
FROM t
WHERE REGEXP(Name.Url, '^http') AND DocId < 20;
```

```
Id: 10                                    t1
Name
  Cnt: 2
  Language
    Str: 'http://A,en-us'
    Str: 'http://A,en'
Name
  Cnt: 0
```

```
message QueryResult {
  required int64 Id;
  repeated group Name {
    optional uint64 Cnt;
    repeated group Language {
      optional string Str; }}}
```

Figure 6: Sample query, its result, and output schema

QUERY的执行

本节描述在数据分布式存储之后，如何尽可能并行的执行计算过程。核心概念就是实现一个树状的执行过程，将服务器分配为树中的逻辑节点，每个层级的节点履行不同的职责，最终完成整个查询。整个过程可以理解成一个任务分解和调度的过程。Query会被分解成多个子任务，子任务调度到某个节点上执行，该节点可以执行任务返回结果到上层的父节点，也可以继续拆解更小的任务调度到下层的子节点。此方案称为服务树（serving-tree）

QUERY的执行

Dremel使用一个多层级服务树来执行查询（见图7）。一个根节点服务器接收到来的查询，从表中读取元数据，将查询路由到下一层。叶子服务器负责与存储层通讯，或者直接在本地磁盘访问数据。

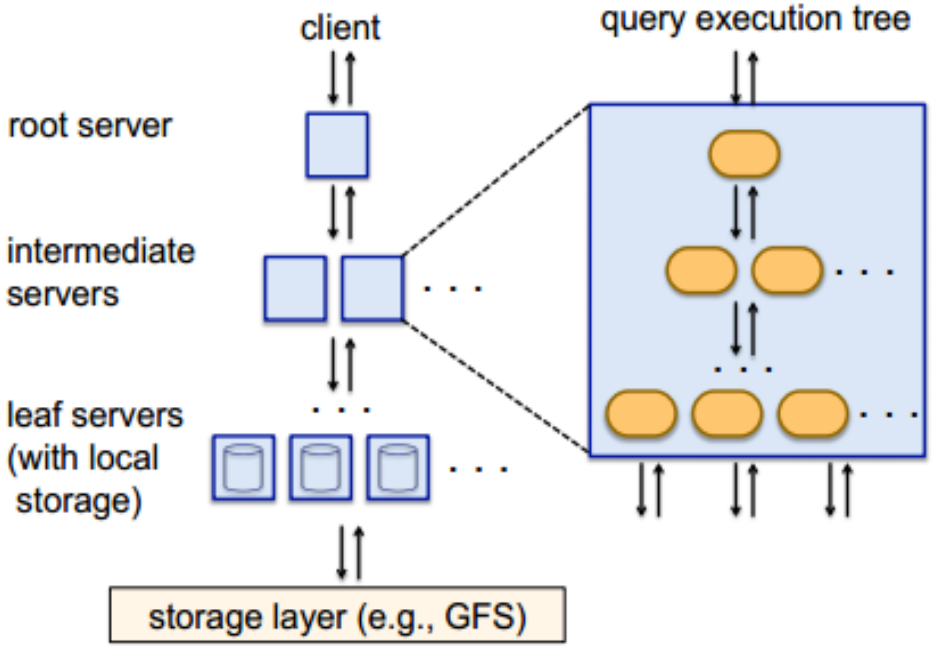


Figure 7: System architecture and execution inside a server node

QUERY的执行

当根节点服务器收到上述查询时，它确定出所有 tablet，也就是表格（把一个 column-stripe 理解成一个 table，称之为 T，table 被分布式存储和查询时可认为 T 进行了水平拆分，tablet 就相当于一个分区）的水平分割，重写查询为如下：

SELECT A,SUM(c) FROM (R_1^1 UNION ALL ... R_n^1) GROUP BY A

R_1^1 到 R_n^1 是树中第 1 层的（1 到 n）节点返回的子查询结果：

R_i^1=SELECT A, COUNT(B) AS c FROM T_i^1 GROUP BY A

每一层的节点所做的都是与此相似的重写（rewrite）过程。查询任务被一级级的分解成更小的子任务，最终落实到叶子节点，并行地对T的tablet进行扫描。在向上返回结果的过程中，中间层的服务器担任了对子查询结果进行聚合的角色。

QUERY的执行

查询分发器。Dremel是一个多用户系统，多个查询通常会被同时执行。一个查询分发器会基于查询任务的优先等级和负载均衡对查询任务进行调度。它还能帮助实现容错机制，当一个服务器变得很慢或者一个tablet备份不可访问时可以重新调度。

查询分发器有一个重要参数，它表示在返回结果之前一定要扫描百分之多少的tablet，设置这个参数到较小的值（比如98%而不是100%）通常能显著地提升执行速度，特别是当使用较小的复制系数时。

小结

本章描述了一种能对大数据进行交互式分析的分布式系统，Dremel。Dremel由几个简单是组件组成，是一种通用的，管理可扩展数据的解决方案，能在短时间内完成对大规模数据的交互式查询与分析。同时Dremel具有很强的可扩展性、稳定性。它实现了对MapReduce的一种互补。

主讲教师和助教

主讲教师：林子雨

单位：厦门大学计算机科学系
E-mail: ziyulin@xmu.edu.cn
个人网页：http://www.cs.xmu.edu.cn/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn

助教：赖明星

单位：厦门大学计算机科学系数据库实验室2011级硕士研究生（导师：林子雨）
E-mail: mingxinglai@gmail.com
个人主页：http://mingxinglai.com

欢迎访问《大数据技术基础》2013班级网站：http://dblab.xmu.edu.cn/node/423
本讲义PPT存在配套教材《大数据技术基础》，请到上面网站下载。

Department of Computer Science, Xiamen University, Sep, 2013