中山大學
SUN YAT-SEN UNIVERSITY

大数据分析的案例、方法与挑战

DTCC2012

2012.4

中山大学海量数据与云计算研究中心 黄志洪

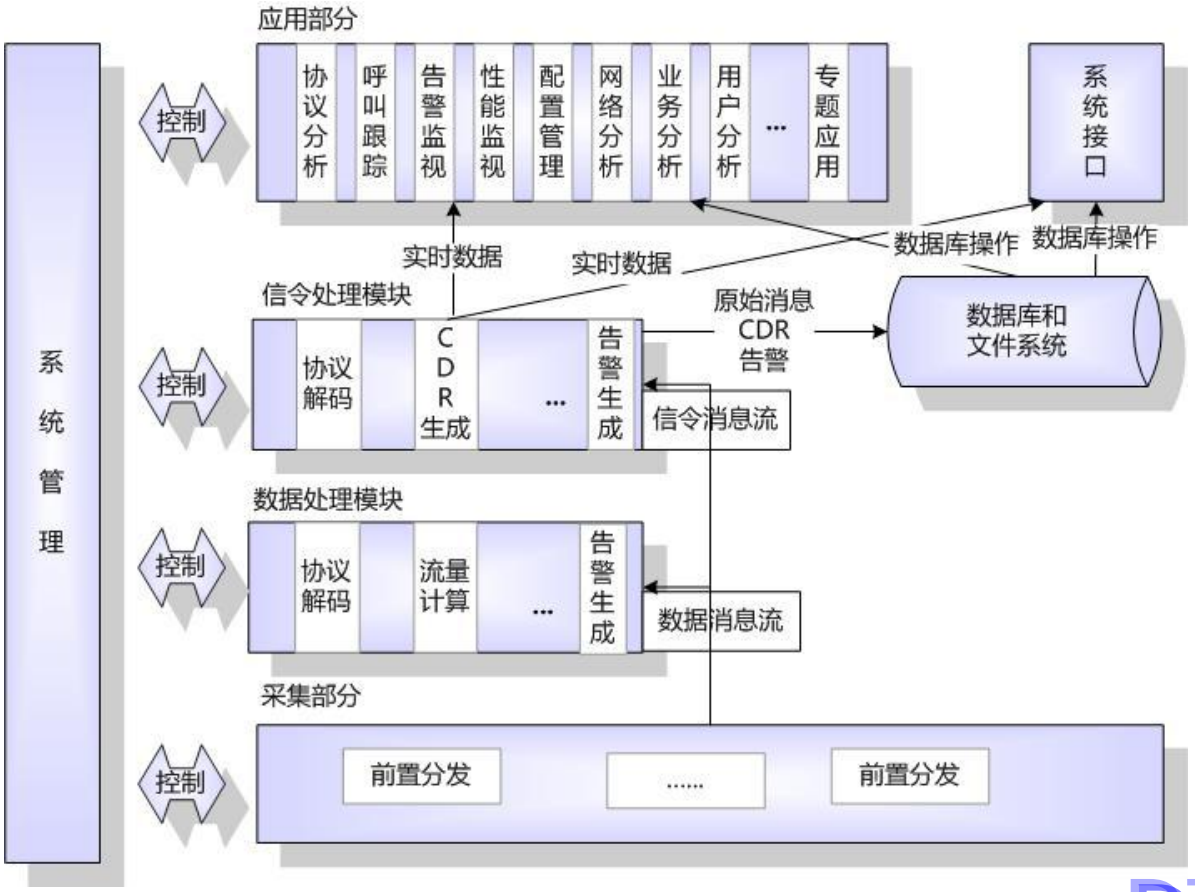
数据分析者面临的问题

中山大學 SUN YAT-SEN UNIVERSITY

- 数据日趋庞大，无论是入库和查询，都出现性能瓶颈
- 用户的应用和分析结果呈整合趋势，对实时性和响应时间要求越来越高
- 使用的模型越来越复杂，计算量指数级上升
- 传统技能无法应对大数据：R、SAS、SQL

场景介绍

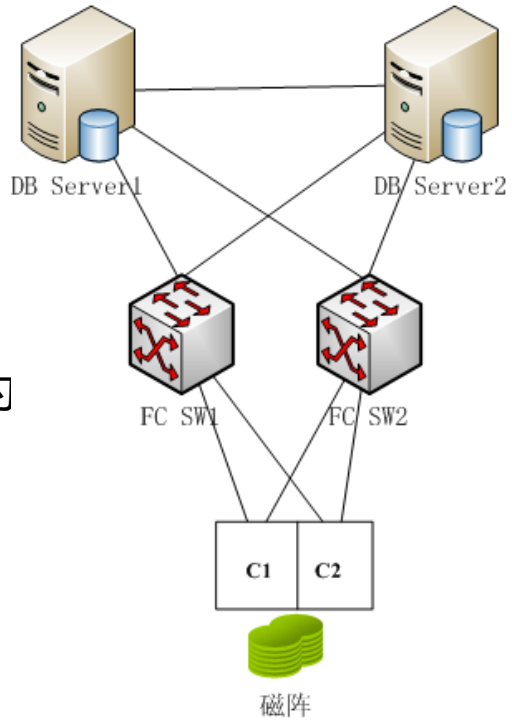
- 信令监测是做什么的？



体系架构

- 数据库服务器：HP小型机，128G内存，48颗CPU，2节点RAC，其中一个节点用于入库，另外一个节点用于查询
- 存储：HP虚拟化存储，>1000个盘
- 入库节点
- 入库方式——常规路径sqlldr
- 大量使用表分区设计
- 数据量：每小时写入200G左右数据磁盘物理写大约为450G每小时

问题：1 **入库瓶颈** 2 **查询瓶颈**



数据库设计

- 物理上采用ASM
- 大表全部按时间分区，开始时按小时分区，但由于数据量庞大，后来改成15分钟分区，最后变成每分钟切换1个分区
- 采用sqlldr方式入库

入库故障描述

- 由于数据量太大，不得不同时启用多个处理机，产生了多个入库节点
- 当入库节点分别增加到2节点和4节点以后，sqlldr出现停顿现象

```
2011-10-15 21:39:01 : 1968288
2011-10-15 21:40:01 : 2276724
2011-10-15 21:41:01 : 2089032
2011-10-15 21:42:01 : 2039904
2011-10-15 21:43:01 : 2925620
2011-10-15 21:44:01 : 4059772
2011-10-15 21:45:01 : 5656632
2011-10-15 21:46:01 : 7126444
2011-10-15 21:47:01 : 8682632
2011-10-15 21:48:23 : 8949184
2011-10-15 21:51:46 : 8075236
2011-10-15 21:51:46 : 8075236
2011-10-15 21:51:56 : 8194924
2011-10-15 21:52:12 : 6999420
2011-10-15 21:53:01 : 6236328
2011-10-15 21:54:01 : 6054180
```

```
11:18:09 PM     IFACE   rxpck/s   txpck/s   rxbyt/s   txbyt/s   rxcmp/s   txcmp/s  rxmcst/s
11:18:12 PM        lo      5.33      5.33    701.67    701.67      0.00      0.00      0.00
11:18:12 PM      eth0  10413.67   3031.67 23056211.33 22577187.33      0.00      0.00      0.00
11:18:12 PM      eth1   3281.33   1958.67 216772.00 13110924.33      0.00      0.00      0.00
11:18:12 PM      eth2      0.00      0.00      0.00      0.00      0.00      0.00      0.00
11:18:12 PM      eth3      0.00      0.00      0.00      0.00      0.00      0.00      0.00
11:18:12 PM      sit0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
```

```
03:51:37 PM     IFACE   rxpck/s   txpck/s   rxbyt/s   txbyt/s   rxcmp/s   txcmp/s  rxmcst/s
03:51:40 PM        lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00
03:51:40 PM      eth0   6782.61   1934.78 14625741.81 14365918.39      0.00      0.00      0.00
03:51:40 PM      eth1      1.34      1.00    263.55     66.22      0.00      0.00      0.00
03:51:40 PM      eth2      0.00      0.00      0.00      0.00      0.00      0.00      0.00
03:51:40 PM      eth3      0.00      0.00      0.00      0.00      0.00      0.00      0.00
03:51:40 PM      sit0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
```

AWR报告

中山大學 SUN YAT-SEN UNIVERSITY

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	99.93	Redo NoWait %:	100.00
Buffer Hit %:	99.96	In-memory Sort %:	100.00
Library Hit %:	100.00	Soft Parse %:	100.00
Execute to Parse %:	12.34	Latch Hit %:	97.39
Parse CPU to Parse Elapsd %:	91.42	% Non-Parse CPU:	99.96

Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
buffer busy waits	1,293,015	175,497	136	62.38	Concurrency
DB CPU		53,377		18.97	
db file sequential read	1,024,484	10,679	10	3.80	User I/O
enq: TX - contention	17,754	10,573	596	3.76	Other
log file sync	188,881	10,158	54	3.61	Commit

AWR报告

Foreground Wait Class

- s - second, ms - millisecond - 1000th of a second
- ordered by wait time desc, waits desc
- %Timeouts: value of 0 indicates value was < .5%. Value of null is truly 0
- Captured Time accounts for 100.2% of Total DB time 281,330.13 (s)
- Total FG Wait Time: 228,626.02 (s) DB CPU time: 53,377.39 (s)

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	%DB time
Concurrency	5,075,450	0	178,342	35	63.39
DB CPU			53,377		18.97
Other	1,368,410	2	11,741	9	4.17
User I/O	1,051,181	0	10,758	10	3.82
Commit	188,881	0	10,158	54	3.61
Configuration	82,543	2	9,725	118	3.46
Cluster	1,284,938	0	6,223	5	2.21
Network	21,152,937	0	1,680	0	0.60
System I/O	72	0	0	3	0.00
Application	159	0	0	0	0.00
Administrative	3	0	0	1	0.00

AWR报告

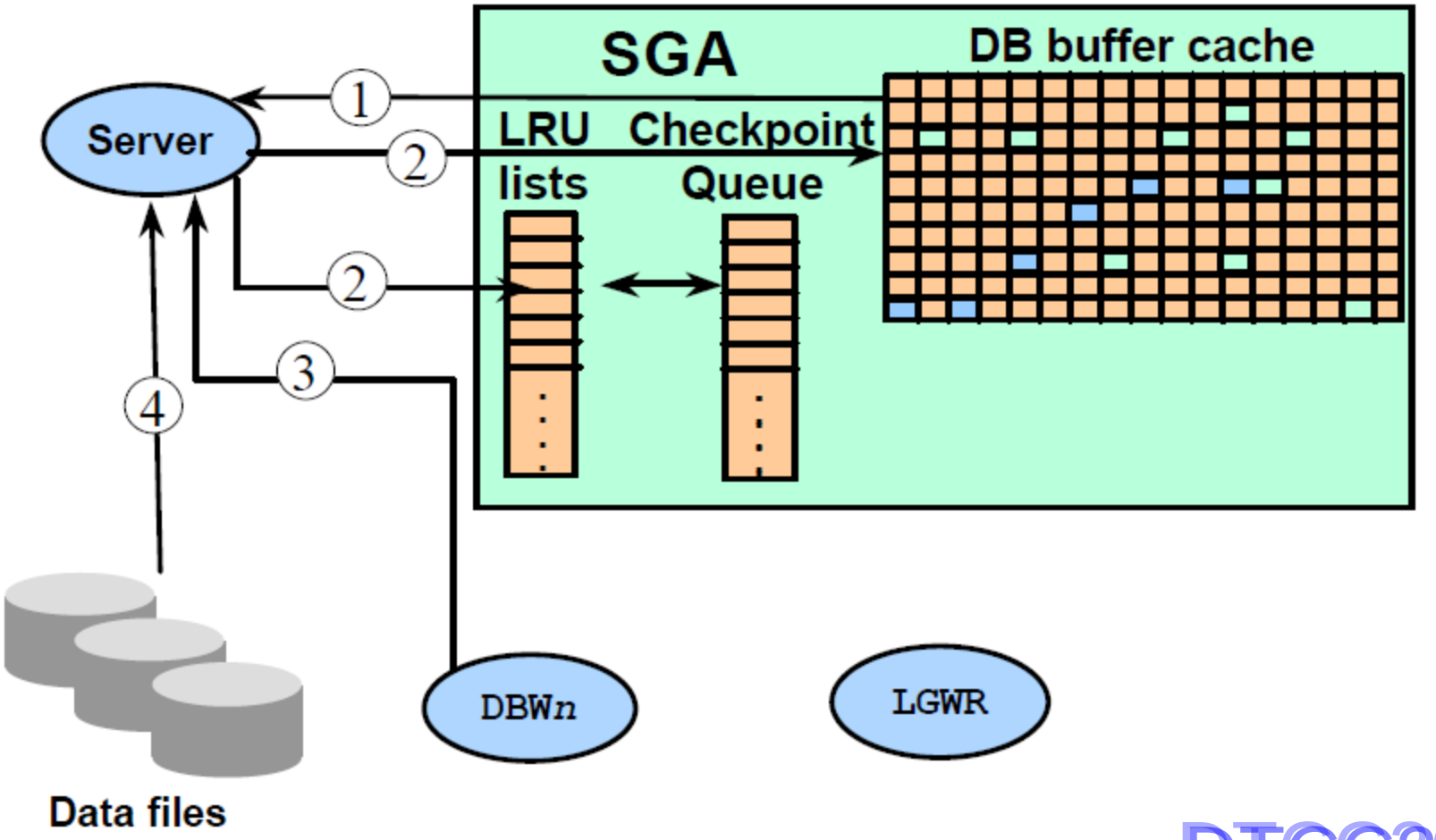
中山大學 SUN YAT-SEN UNIVERSITY

Foreground Wait Events

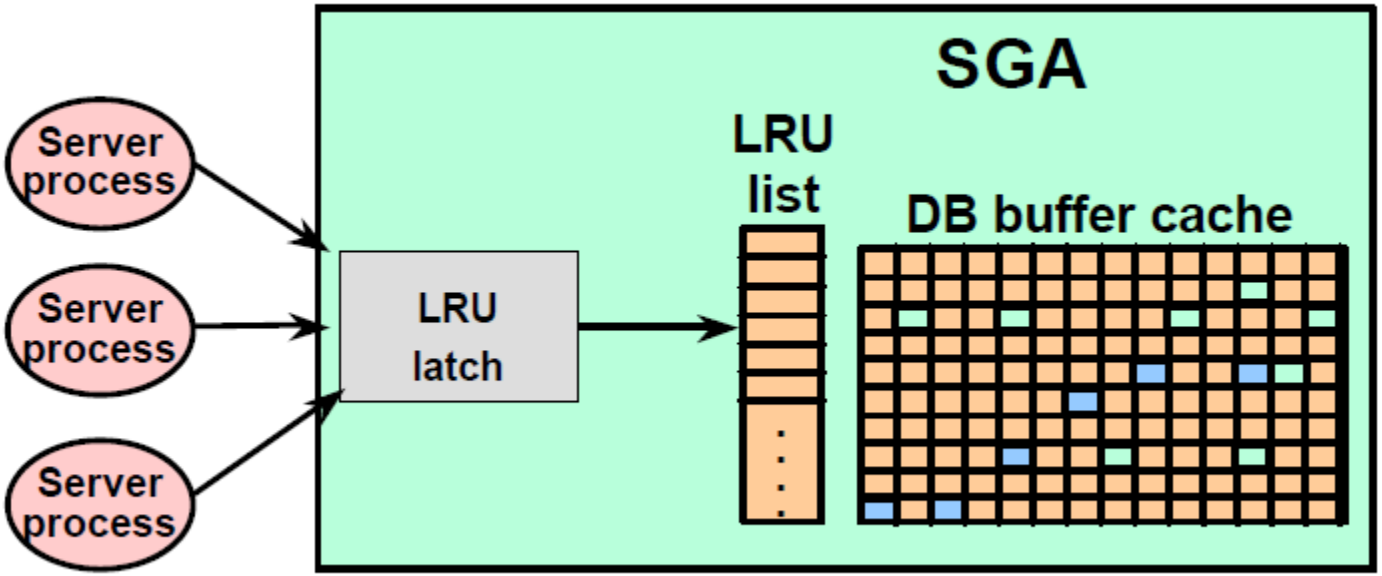
- s - second, ms - millisecond - 1000th of a second
- Only events with Total Wait Time (s) >= .001 are shown
- ordered by wait time desc, waits desc (idle events last)
- %Timeouts: value of 0 indicates value was < .5%. Value of null is truly 0

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn	% DB time
buffer busy waits	1,293,015	0	175,497	136	6.65	62.38
db file sequential read	1,024,484	0	10,679	10	5.27	3.80
enq: TX - contention	17,754	0	10,573	596	0.09	3.76
log file sync	188,881	0	10,158	54	0.97	3.61
enq: HW - contention	61,795	0	5,794	94	0.32	2.06
gc buffer busy release	68,848	0	4,003	58	0.35	1.42
log buffer space	18,416	0	2,702	147	0.09	0.96
latch: cache buffers chains	3,774,137	0	1,965	1	19.40	0.70
SQL*Net more data from client	18,938,834	0	1,678	0	97.35	0.60
undo segment extension	1,727	98	1,151	666	0.01	0.41
latch: redo allocation	1,108,775	0	858	1	5.70	0.30
gc current block 2-way	786,322	0	691	1	4.04	0.25
enq: TX - index contention	3,854	0	485	126	0.02	0.17
gc current grant busy	92,682	0	473	5	0.48	0.17

关于Buffer Cache



Latch


SGA
LRU
list
DB buffer cache
Server
process
Server
process
Server
process
LRU
latch

寻找Buffer busy wait的根源

中山大學 SUN YAT-SEN UNIVERSITY

- Sqlldr和OCI方式同时insert
- 多个节点同时insert
- 解决办法

1 放弃使用OCI

2 对sqlldr进行垂直切分，尽量避免同时多进程插入同一张表

再看AWR

中山大學 SUN YAT-SEN UNIVERSITY

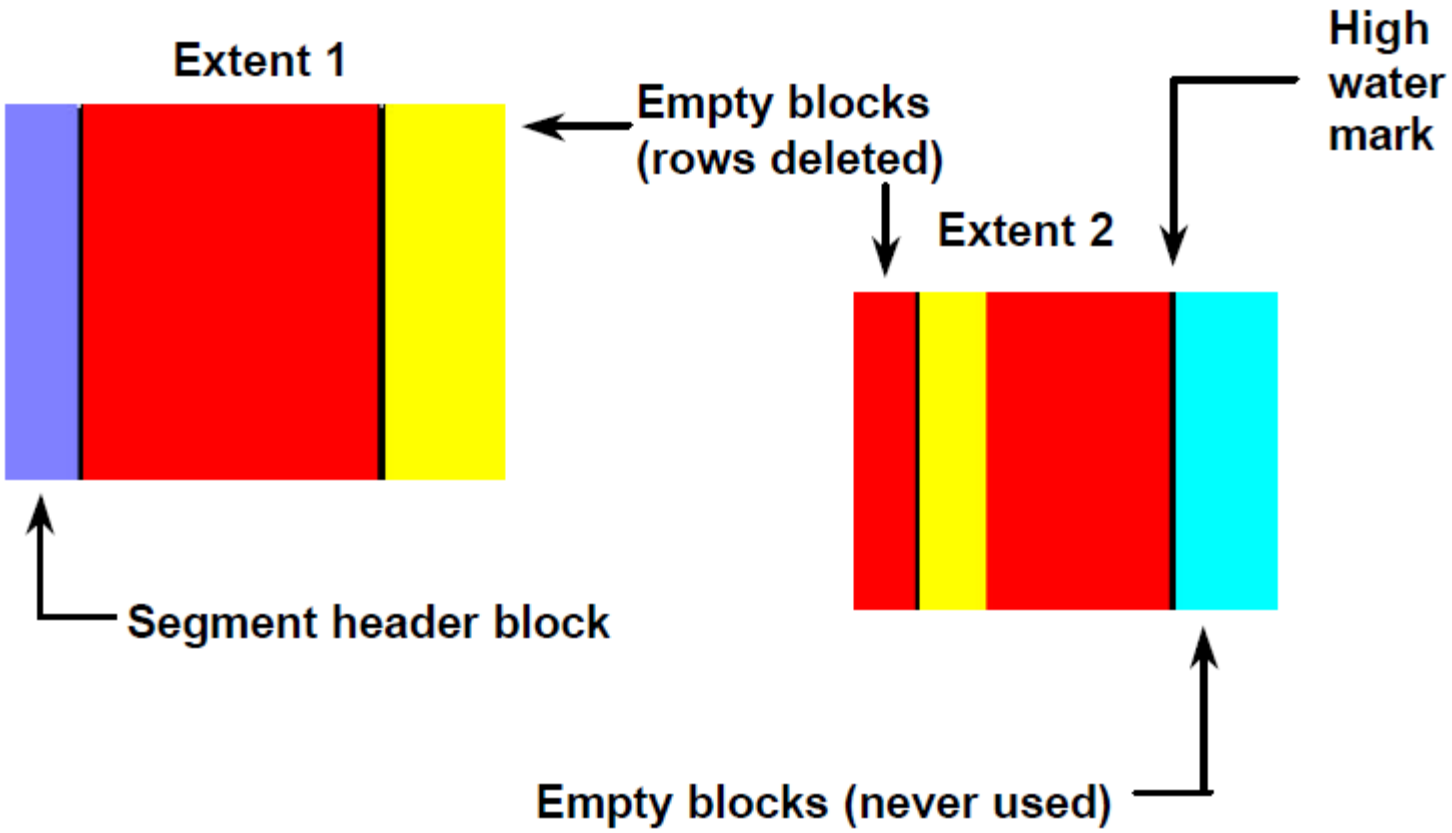
Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
DB CPU		23,955		74.03	
enq: TX - contention	4,322	2,521	583	7.79	Other
buffer busy waits	347,881	1,946	6	6.01	Concurrency
db file sequential read	246,785	1,605	7	4.96	User I/O
enq: HW - contention	13,186	964	73	2.98	Configuration

新的故障现象

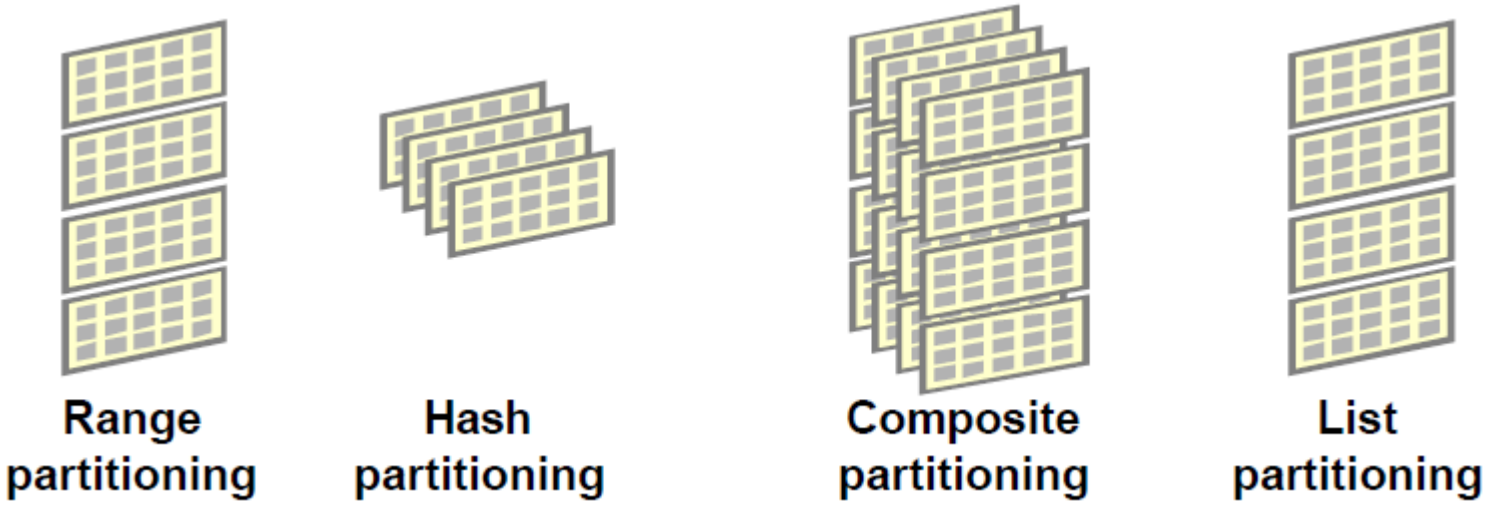
- Sqlldr依然有停顿，次数较为频密而持续时间较短
- HWM冲突问题

关于HWM


Extent 1
Empty blocks
(rows deleted)
High
water
mark
Extent 2
Segment header block
Empty blocks (never used)

针对HWM冲突的优化措施

- 对于无法垂直切分的特大表，按照入库节点号作子分区



再看AWR

- HWM冲突已经被消除
- Sqlldr频密周期性短暂停顿的问题依旧

Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
DB CPU		22,295		67.72	
buffer busy waits	246,775	7,349	30	22.32	Concurrency
db file sequential read	170,318	970	6	2.95	User I/O
enq: TX - contention	1,355	918	677	2.79	Other
SQL*Net more data from client	10,857,139	828	0	2.51	Network

最终问题根源

- AWR报告的提示——文件头部竞争
- 表空间大小与自动扩展是问题根源
- 修正表空间设置后问题消失

Buffer Wait Statistics

- ordered by wait time desc, waits desc

Class	Waits	Total Wait Time (s)	Avg Time (ms)
file header block	20,342	7,320	360
1st level bmb	149,160	13	0
undo header	92,592	12	0
data block	22,804	4	0
segment header	1,271	0	0
2nd level bmb	380	0	0

备选方案——牺牲实时性换取直接路径插入

- 直接路径插入有什么好处？
- 为什么没有采用直接路径插入？

IOStat by Function summary

- 'Data' columns suffixed with M,G,T,P are in multiples of 1024 other columns suffixed with K,M,G,T,P are in multiples of 1000
- ordered by (Data Read + Write) desc

Function Name	Reads: Data	Reqs per sec	Data per sec	Writes: Data	Reqs per sec	Data per sec	Waits: Count	Avg Tm(ms)
DBWR	8M	0.14	.002213	214.9G	974.74	60.8853	520	0.07
LGWR	2M	0.05	.000553	184.6G	105.23	52.2994	114.7K	5.23
Others	223M	3.80	.061689	52.1G	15.91	14.7484	15.9K	0.52
Buffer Cache Reads	5.2G	47.25	1.47666	0M	0.00	0M	170.8K	5.26
Direct Writes	0M	0.00	0M	1M	0.00	.000276	0	
TOTAL:	5.4G	51.23	1.54112	451.6G	1095.88	127.933	301.8K	4.99

备选方案——交换分区

```
LOCK TABLE interval_sales
PARTITION FOR (TO_DATE('01-JUN-2007','dd-MON-yyyy'))
IN SHARE MODE;

ALTER TABLE interval_sales
EXCHANGE PARTITION FOR (TO_DATE('01-JUN-2007','dd-MON-yyyy'))
WITH TABLE interval_sales_jun_2007
INCLUDING INDEXES;
```

备选方案——外部表

```
SYS@AS SYSDBA> insert /*+ parallel(x,16) append */into BIP_FUSION.test x select /*+ parallel(t,16) */* from BIP_FUSION.OG_USER_D01_EXTEST t;

65315447 rows created.

Elapsed: 00:00:45.14
```

使用传统关系型数据库遇到的困难

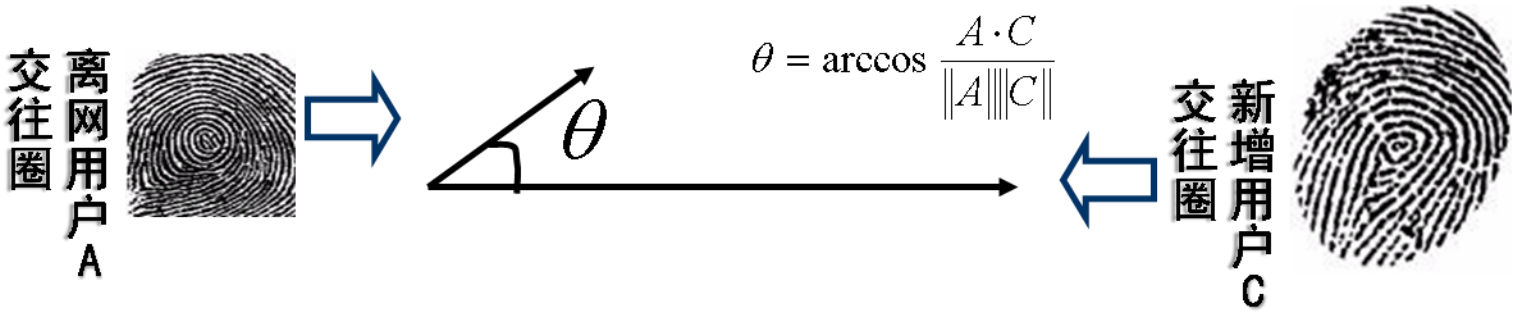
- All – in – one，并非专门针对数据分析设计和优化
- 设计复杂，调优复杂，数据分析师兼任DBA
- 当数据规模增加时，需要扩展硬件，边际成本指数级上升，存在无法突破的物理瓶颈

解决方案

- 列式数据库，实时数据库等新的数据库技术
- 分布式集群：Hadoop，NoSQL及其它分布式数据库技术
- 混合使用各种专业分析产品

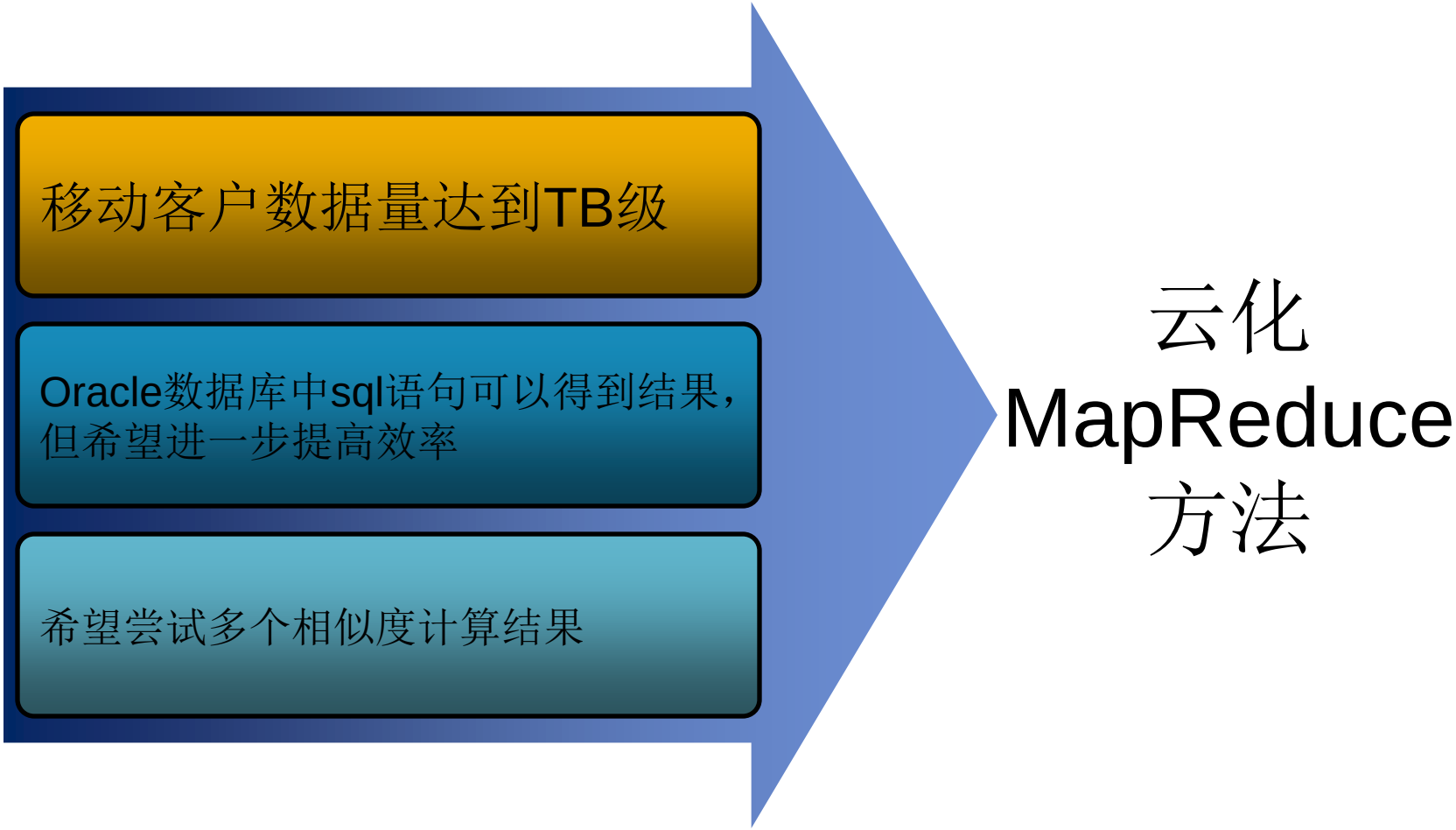
场景：行为指纹识别

$$sim(x,y)=\cos(x,y)=\frac{(x,y)}{\|x\|\cdot\|y\|}=\frac{\sum_{i=1}^{n}x_i\cdot y_i}{\left(\sum_{i=1}^{n}x_i^2\cdot\sum_{i=1}^{n}y_i^2\right)^{1/2}}$$

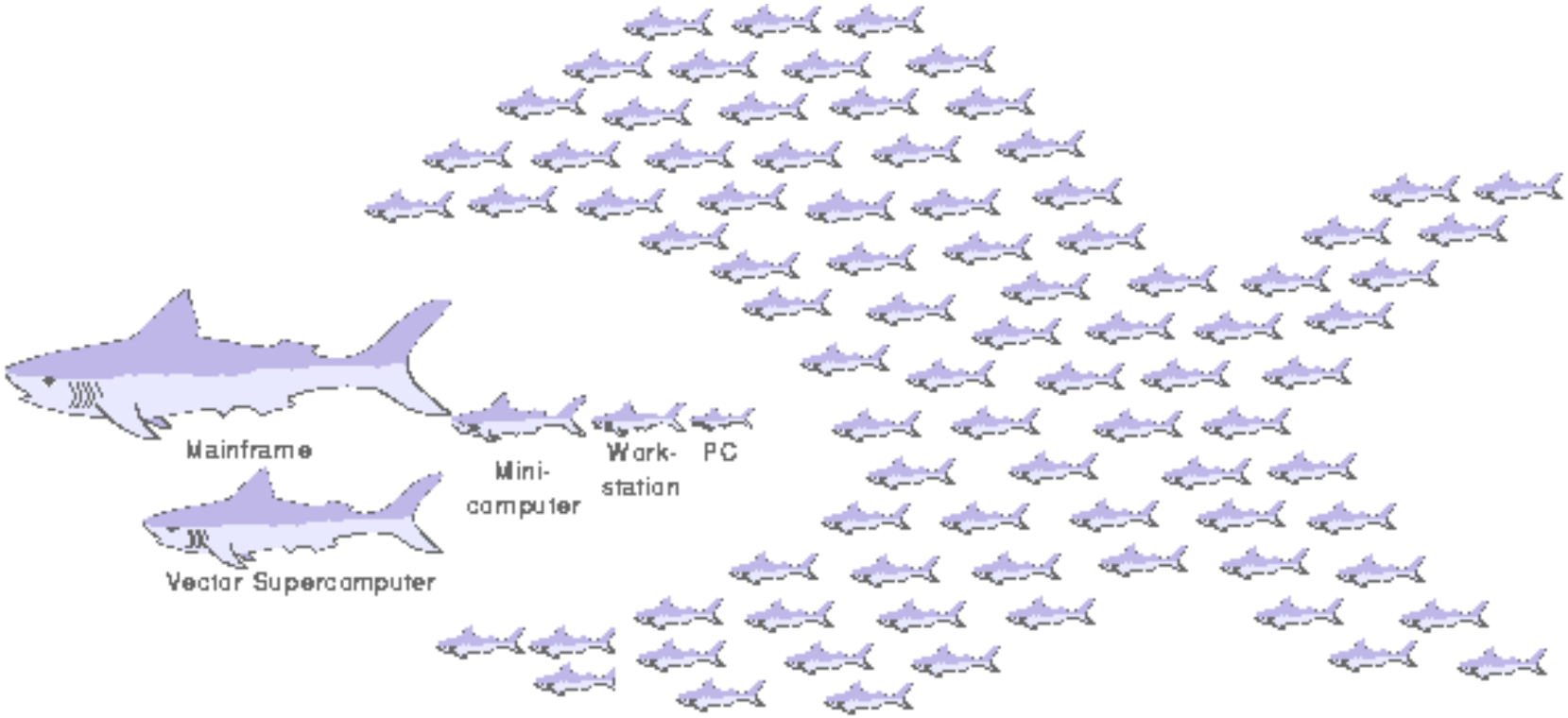


- 当 θ 为 90° 时，AC两个矢量完全不相关，即两个号码的交往圈相似度最低
- 当 θ 为0 时，AC两个矢量完全相关，即两个号码的交往圈相似度最高
- 当 θ 越接近0，说明两个号码的交往圈越相似

基于分布式平台运行海量数据



云计算——网络发展的必然结果



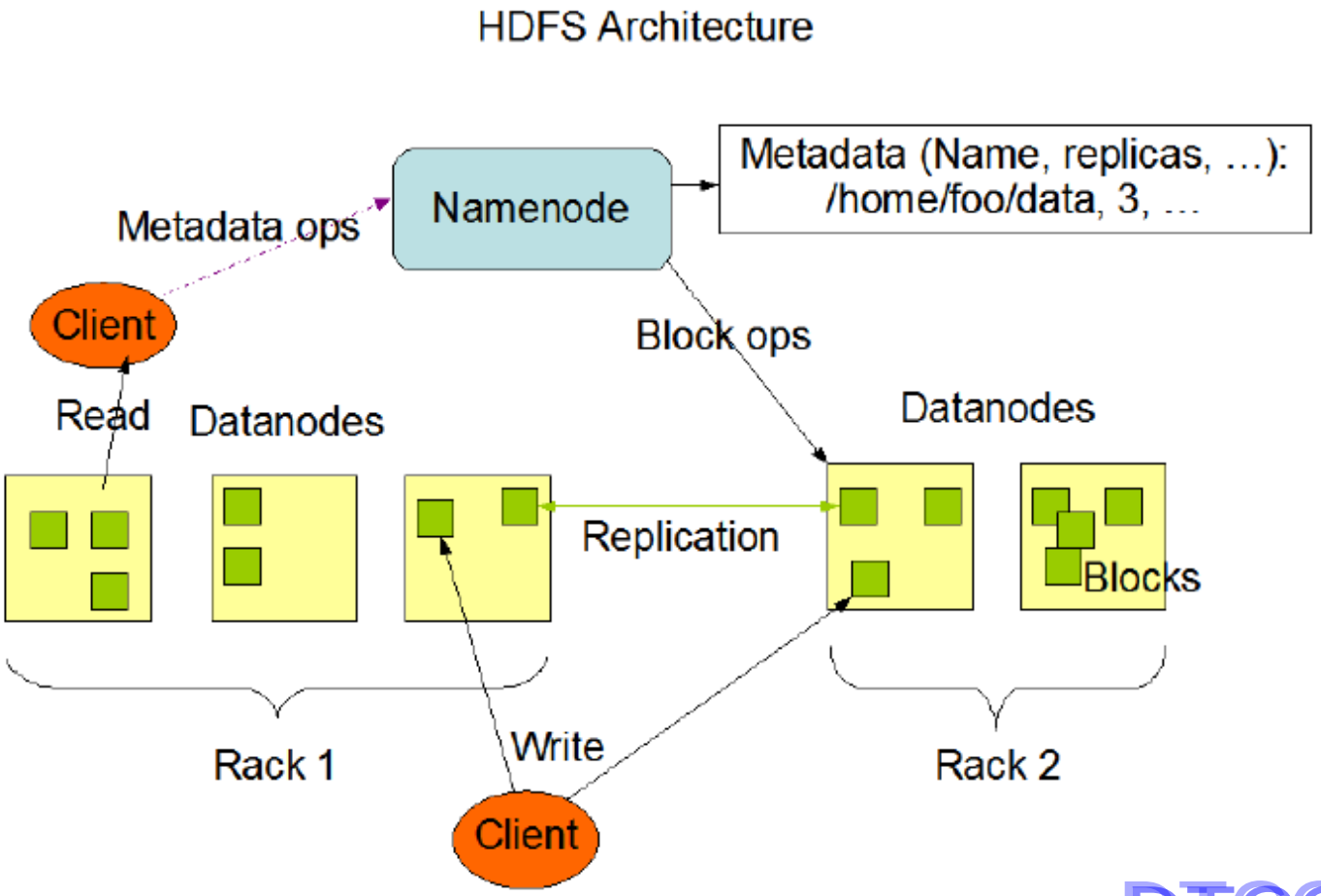
改造

DTCC2012

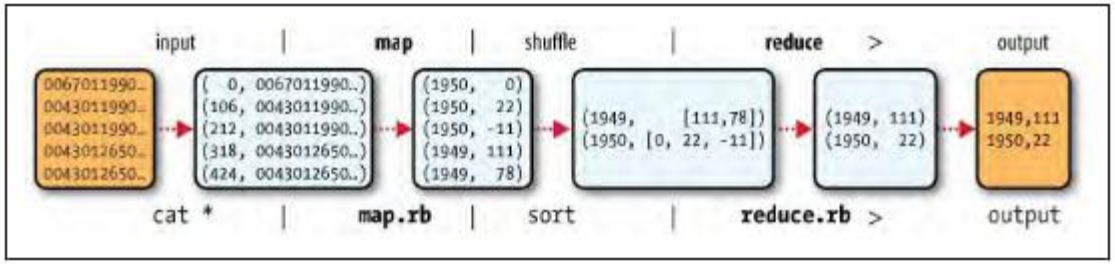
Hadoop

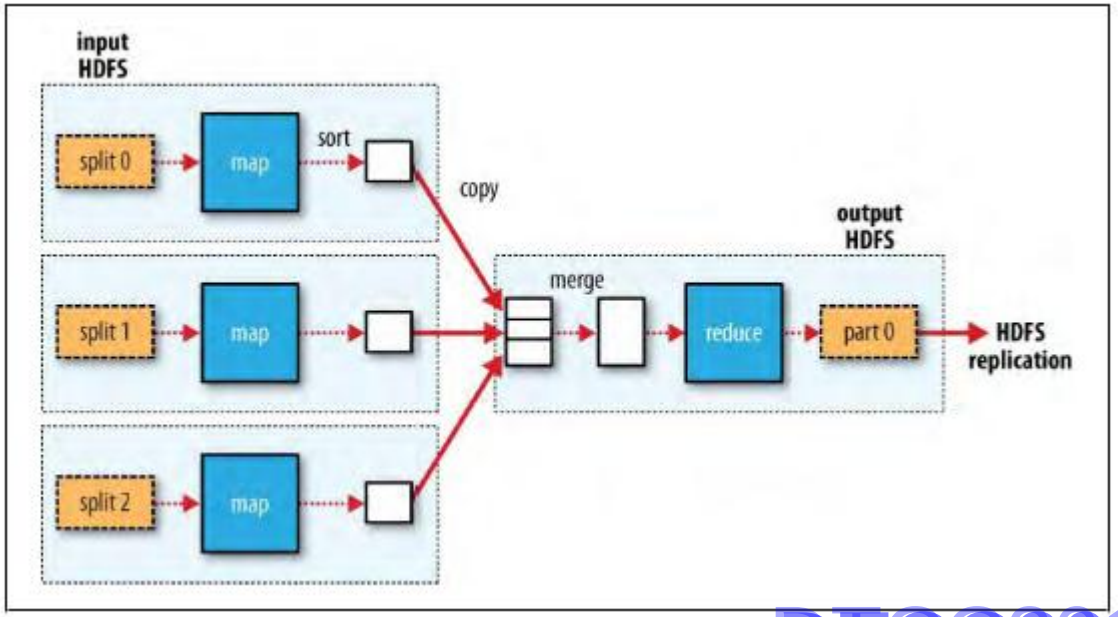
- Hadoop的主要功能：HDFS和Map-Reduce
- HDFS实现数据的分布式存储，并且实现冗余备份
- Map-Reduce实现计算任务的分布化，尽量使到某个节点的计算任务主要面对存储在本地的数据，以减少跨节点的网络数据传送

HDFS结构示意图

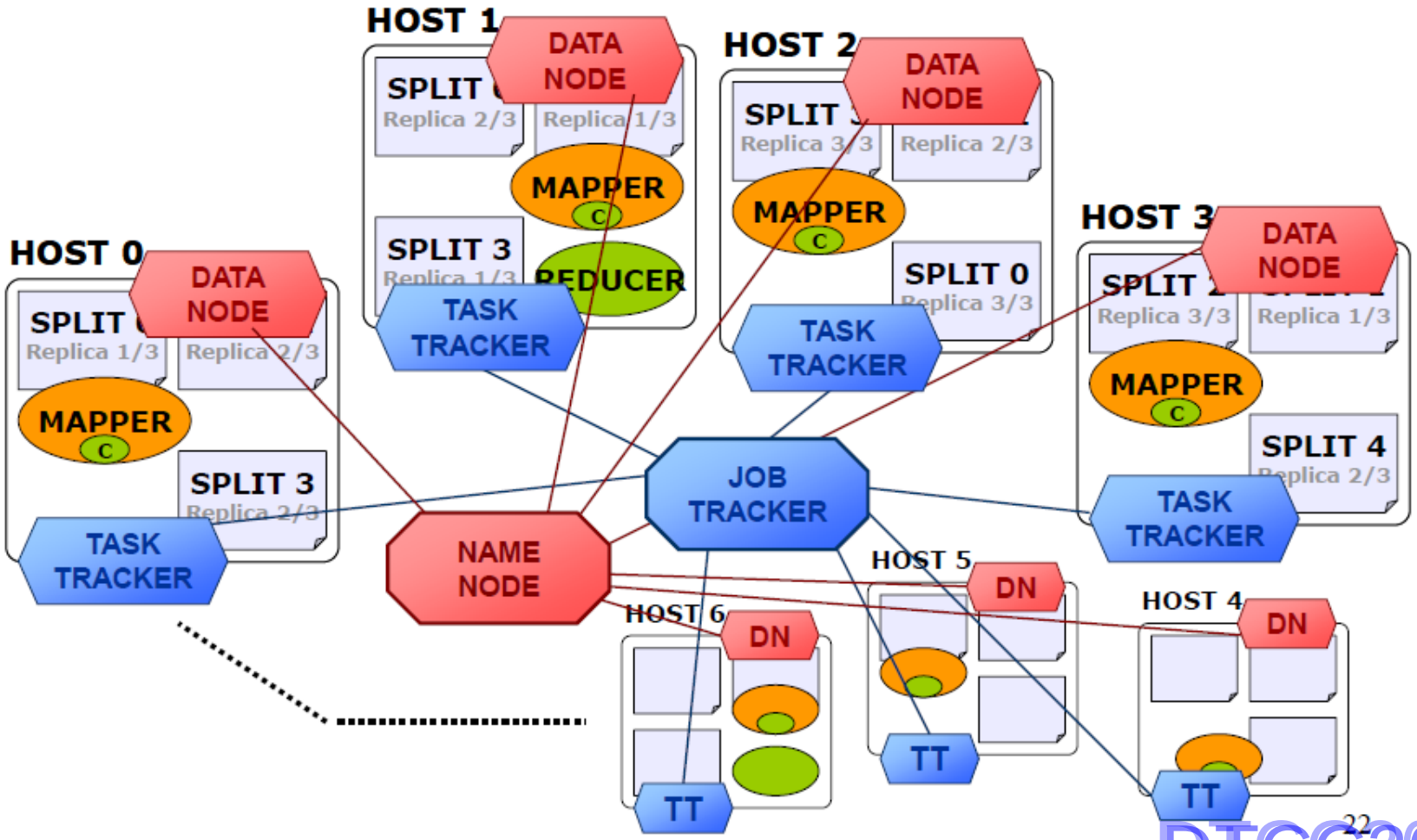

HDFS Architecture
Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

Map-Reduce示意图





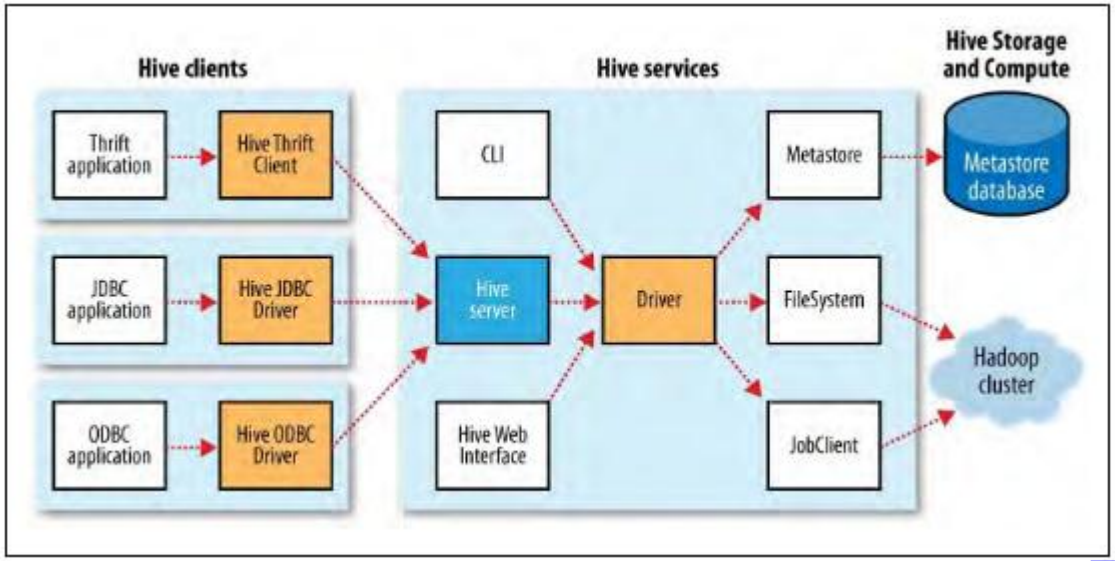
HDFS与Map-Reduce一起工作



2012.4

Hive

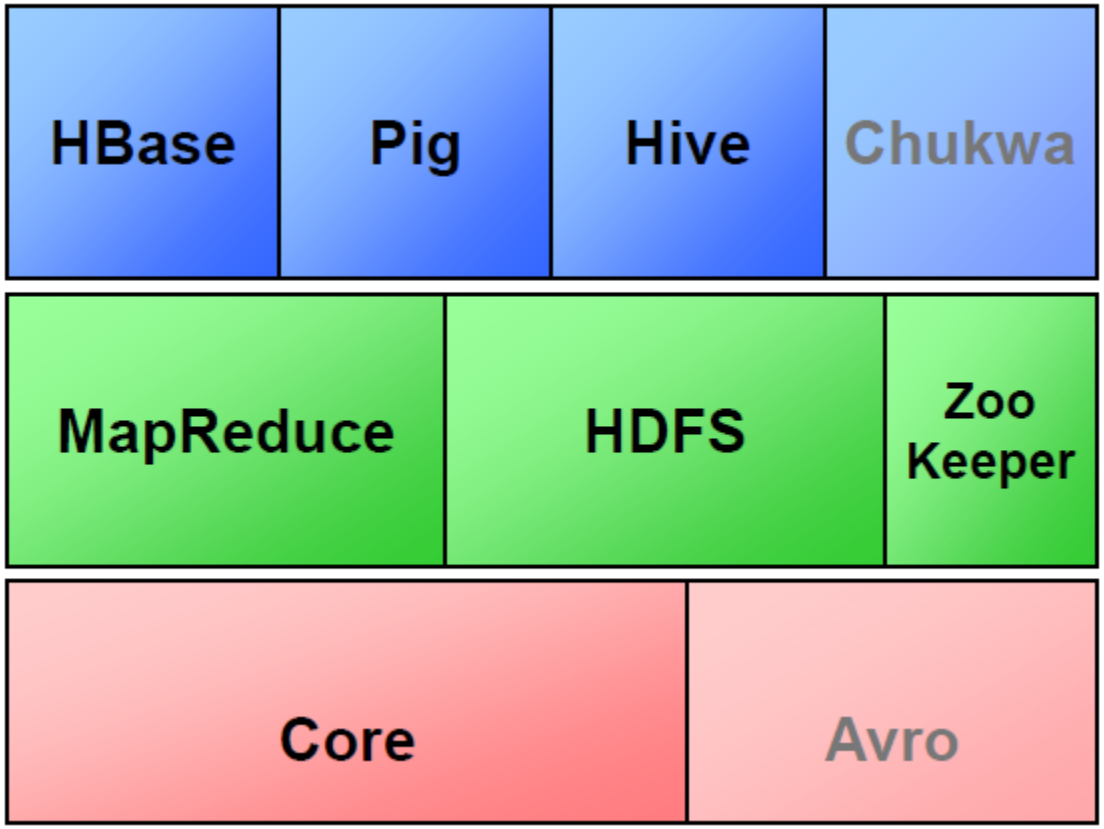
- 基于Hadoop的常用数据分析工具
- 可以看成是SQL到Map-Reduce的转换器
- HiveQL尚未能完全支持SQL 92
- 外部应用可以通过hive客户端、JDBC、ODBC等方式访问Hive



HBase

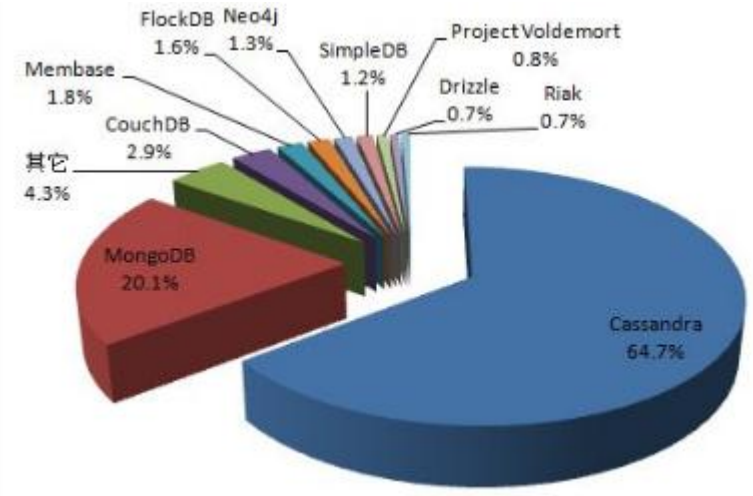
- 列式数据库，特别适合作为数据分析的场景，可以减少I/O
- 无真正索引
- 自动分区
- 增加新节点时自动线性扩展
- 使用Hbase命令而非SQL
- 可以通过Java，REST，thrift等接口访问HBase

Hadoop体系图



NoSQL

- NoSQL = Not Only SQL
- High performance, Huge Storage, High Scalability && High Availability
- NoSQL面向的场景：事务性要求不高，实时性要求不高，查询较为简单，数据海量
- 可分布化，运行在廉价的PC集群上
- 典型的NoSQL产品，通常某种产品只适合某种特定场景，常要配搭使用



基于Hadoop的用户指纹识别算法

- 求某个客户最相似客户的MapReduce化（以相关系数为例）：
- Map（ ）
  - Input： 某客户数值、所有客户数值
  - 将所有客户随机平分到 k台机器
  - Output: k 个最大相关系数(local)
  - Emit the k 个跟某客户最相似的客户
- Reduce( )
  - Input: Key: null; values: k 个最大相关系数(local)
  - Output: 最大相关系数(global)
  - Emit the 最大相关系数、与某客户最相似客户

数据分析者期待的解决方案

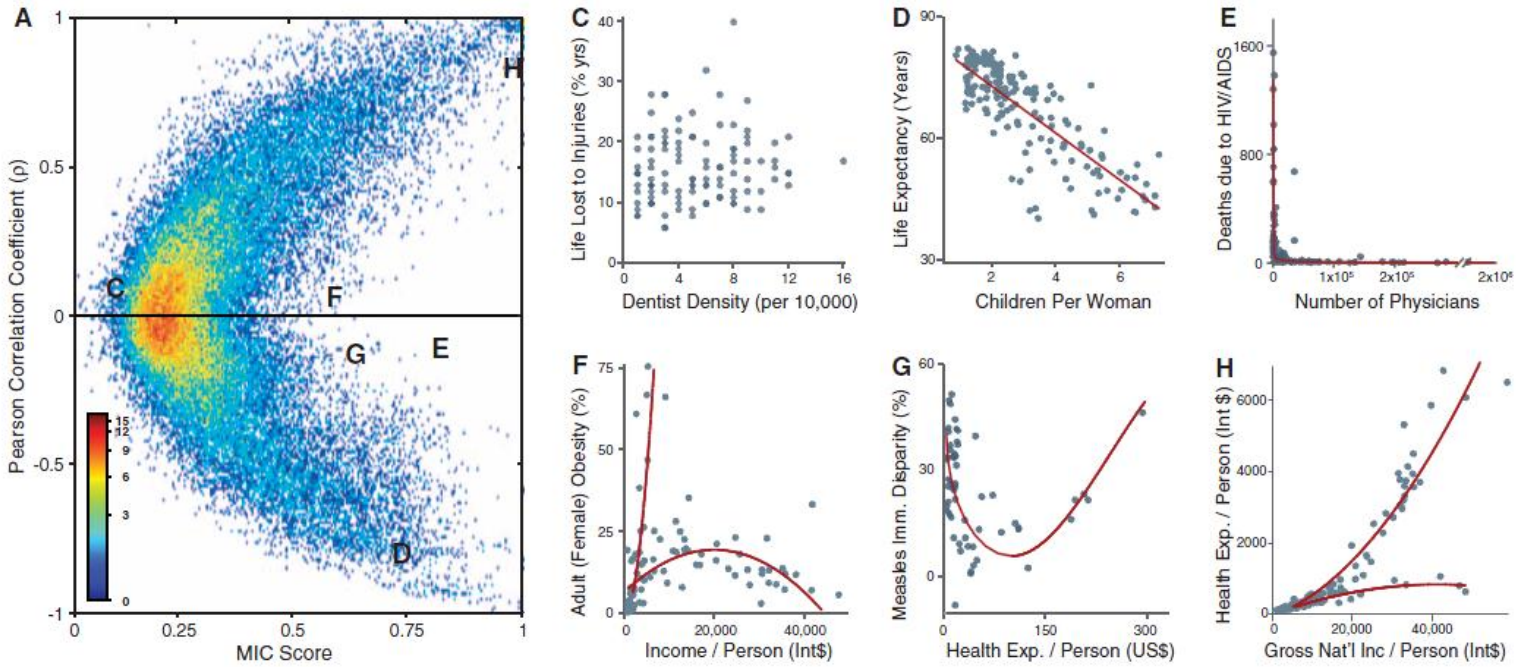
- 完美解决性能瓶颈，在可见未来不容易出现新瓶颈
- 过去所拥有的技能可以平稳过渡。比如SQL、R
- 转移平台的成本有多高？平台软硬件成本，再开发成本，技能再培养成本，维护成本

一种新的普适性关联挖掘方法

- 海量，不一定是指数据记录多，有时可能是变量很多
- 观察变量之间是否具有联系的传统回归方法

传统回归模型的困难

- 为什么一定是线性？或某种非线性模型？
- 过分依赖于分析者的经验
- 对于非连续的离散数据难以处理



网格方法

- 《Science》上的文章《Detecting Novel Associations in Large Data Sets》
- 方法概要：用网格判断数据的集中程度，集中程度意味着是否有关联关系
- 方法具有一般性，即无论数据是怎样分布的，不限于特定的关联函数类型，此判断方法都是有效
- 方法具有等效性，计算的熵值和噪音的程度有关，跟关联的类型无关
- MIC：the Maximal Information Coefficient
- MINE：Maximal Information-based Nonparametric Exploration

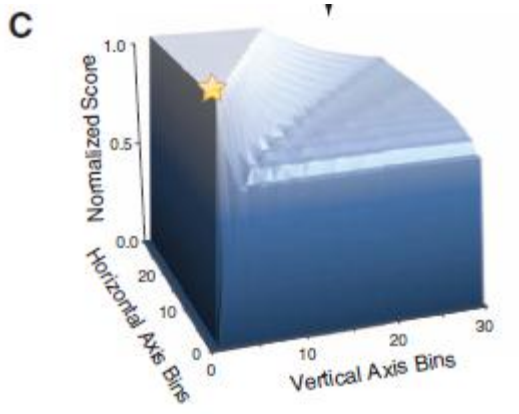
MIC值计算

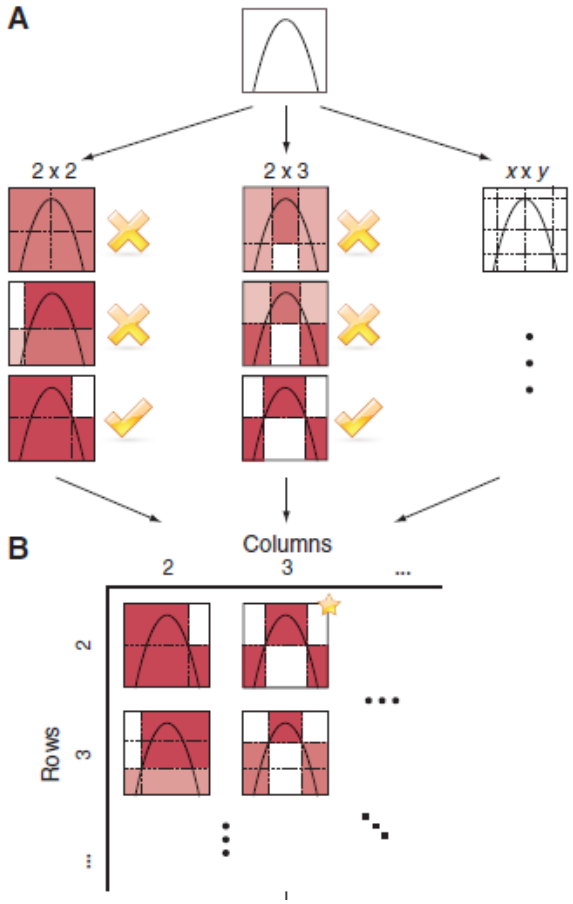
- 坐标平面被划分为(x,y)网格G（未必等宽），其中$xy<n^{0.6}$
- 在G上可以诱导出“自然概率密度函数”p(x,y)，任何一个方格（box）内的概率密度函数值为这个方格所 包含的样本点数量占全体样本点的比例
- 计算网格划分G下的 **mutual information值** I_G

$$I(X;Y) = \int_Y \int_X p(x,y) \log \left(\frac{p(x,y)}{p(x)\,p(y)} \right) dx\, dy$$

MIC值计算

- 构造**特征矩阵**{m_{xy}}，矩阵的元素m_{xy}=max{I_G}/log min{x,y}。max取遍所有可能的(x,y)网格G
- MIC=max {m_{xy}}。Max取遍所有可能的(x,y)对





MIC值计算

- Mxy的计算是个难点，数据科学家构造了一个近似的逼近算法以提高效率

http://www.sciencemag.org/content/suppl/2011/12/14/334.6062.1518.DC1

在作者的网站上，可以下载MINE计算MIC的程序（Java和R）以及测试用数据集

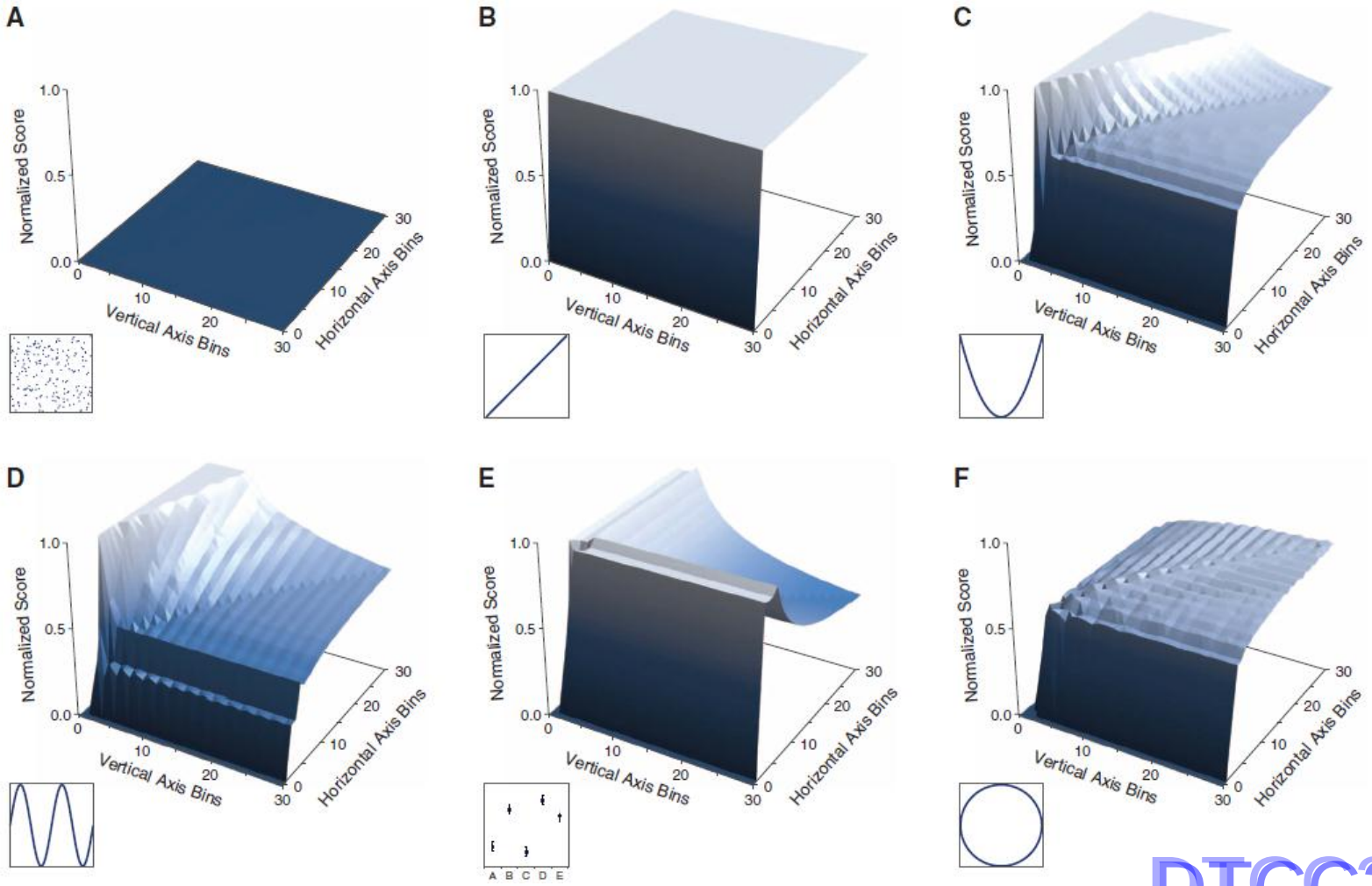
http://www.exploredata.net/Downloads

实验：WHO数据集，垒球数据集…

MIC的性质

- 如果变量对x,y存在函数关系，则当样本数增加时，MIC必然趋向于1
- 如果变量对x,y可以由参数方程c(t)=[x(t),y(t)]所表达的曲线描画，则当样本数增加时，MIC必然趋于1
- 如果变量对x,y在统计意义下互相独立，则当样本数增加时，MIC趋于0

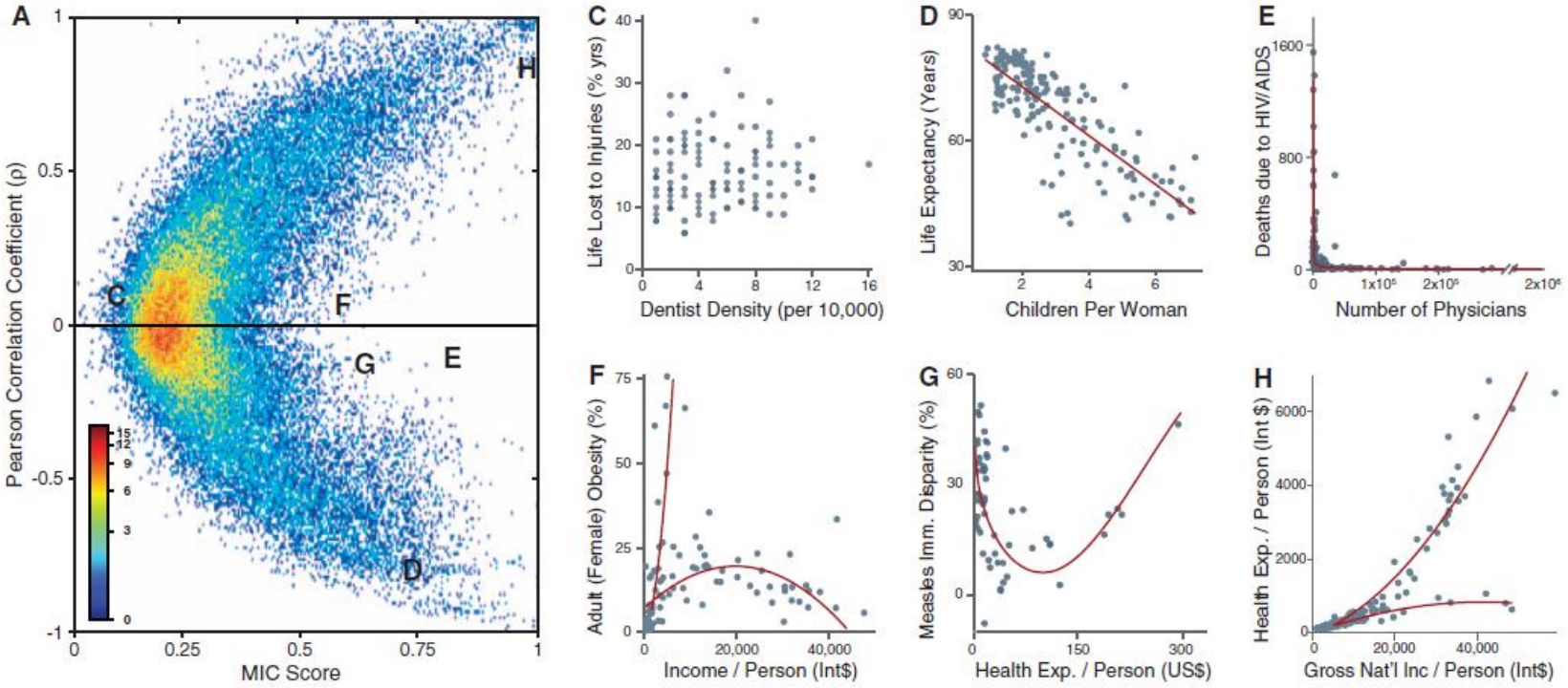
MIC观察



2012.4

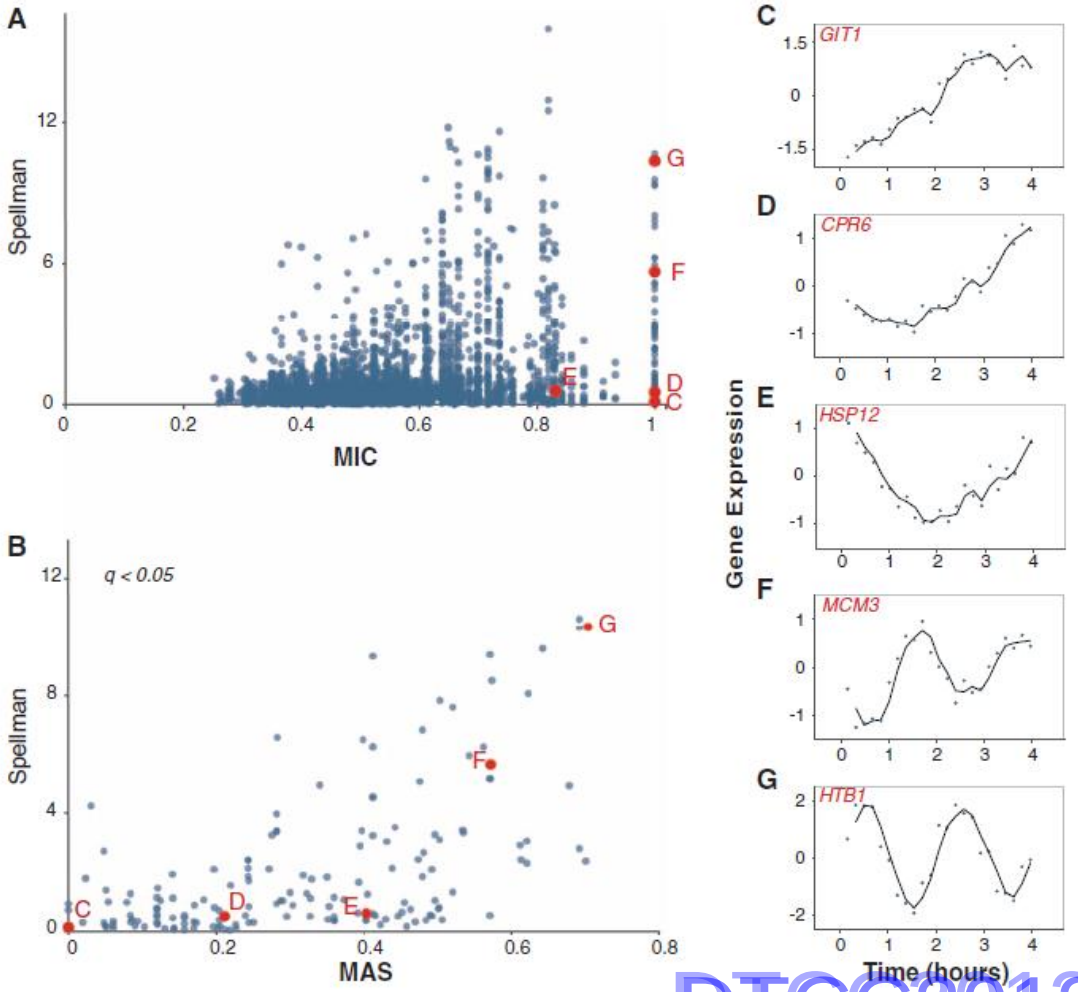
DTCC2012

MIC与线性回归模型对比



对基因数据集spellman的探索

- 数据集包含6223组基因数据
- MINE对关联关系的辨认力明显强于以往的方法，例如双方都发现了HTB1，但MINE方法挖出了过去未被发现的HSP12



2012.4

中山大學
SUN YAT-SEN UNIVERSITY

Thanks

欢迎交流：**stswzh@sysu.edu.cn**

QQ：1829118　　微博：http://weibo.com/haolan2011

DTCC2012