集算器

创新大数据计算引擎

高性能计算专家（HPC）

润乾软件出品

高性能计算理念

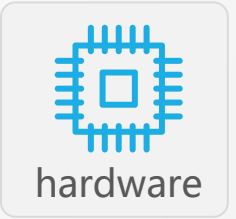
软件改变不了硬件的计算性能

但软件可以设计一些高效（低复杂度）算法，降低计算规模，从而提升计算性能

这类方案通常需要定制化，因地制宜，看菜吃饭

集算器使用了诸多高效算法：

- ✓ 分组计算
- ✓ 关联计算
- ✓ 查找定位
- ✓ 遍历技术
- ✓ 游标与管道
- ✓ 并行计算
- ✓ ……



高效算法举例 – 分组计算

分组计算

结果集小

能够全内存

无需磁盘交互生成临时缓存 • **可**并行

结果集大

不能全内存

针对分组字段有序

无需磁盘交互生成临时缓存 • **可**并行

针对分组字段无序

需要磁盘交互生成临时缓存 • **无法**并行

了解了计算和数据特征，就可以选用适合的算法，从而获得高性能

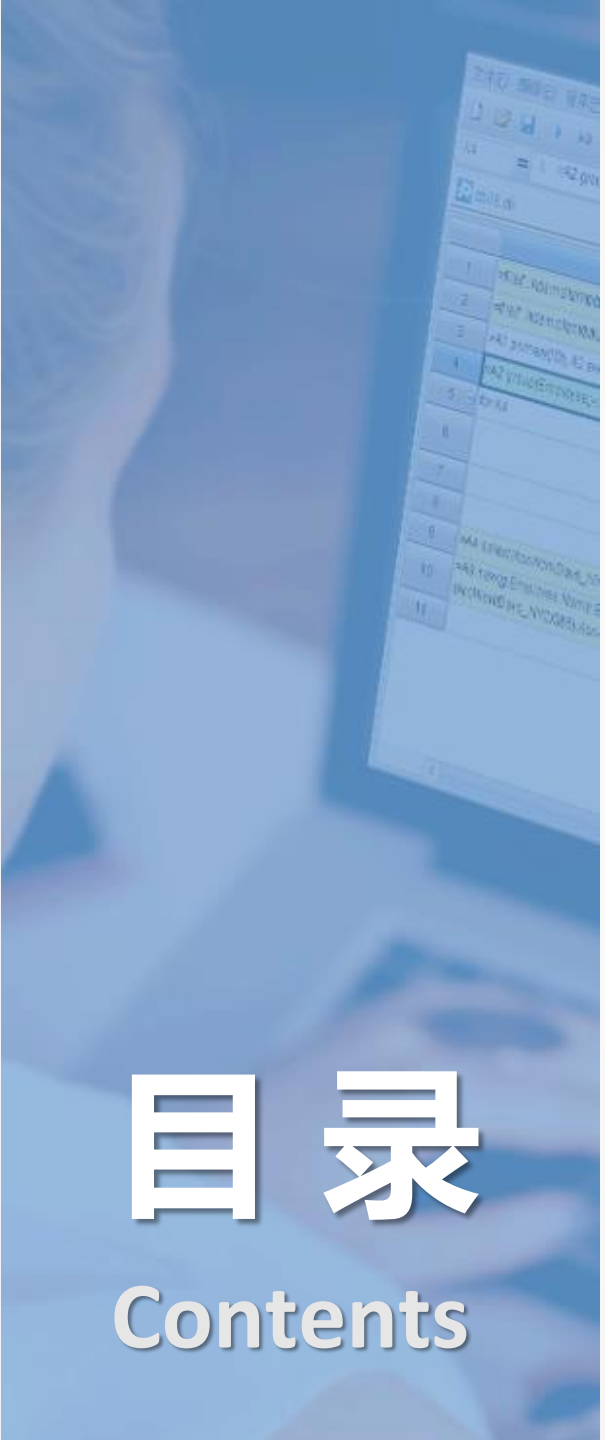
集算器性能表现

*数据规模：100亿行；集群数量：4

测试结果（时间单位：秒）					
测试用例	Intel X86芯片			国产飞腾芯片	
	SPL读文件计算	SPL读数据库计算	数据库中SQL计算	SPL读文件计算	SPL读数据库计算
连接后并集	-	3.8	>1小时	-	-
连接后交集	-	3.9	>1小时	-	-
多对多连接遍历	**69**	103	>1小时	**93**	268
有序分组遍历	**100**	647	>1小时	**102**	2037
多步过程计算	**272**	848	>1小时	**377**	>1小时
大分组	**39**	155	2573	**56**	2493
大表关联分组	**111**	566	>1小时	**178**	2106
批量键值查询	**15**	>1小时	>1小时	**15**	>1小时

【注】SPL是润乾集算器采用的程序设计语言；SQL是关系数据库采用的程序设计语言

国产飞腾芯片上运行的润乾集算器可以超越Intel芯片上分布式数据库的性能



目录

Contents

沉重的大数据计算

1

集群

单机性能不被关注，

依靠集群规模

2

内存

避开外存计算困难，

指望巨大内存

3

框架

框架体系庞大复杂，

试图包罗万象

大数据计算开发难度大

大数据平台对SQL查询关注过多

性能比拼的主要阵地

优化SQL性能几乎无助于降低开发难度

大量过程计算的开发难度还很大

用SQL很难描述，复杂SQL优化效果不好

仍需大量底层的编码，经常编写UDF

提高性能本质上是降低开发难度

复杂运算的自动优化靠不住，需要快速编写高性能算法

集群透明化

大数据平台努力实现集群透明化

单机与集群一致

网络存储系统+自动任务分配

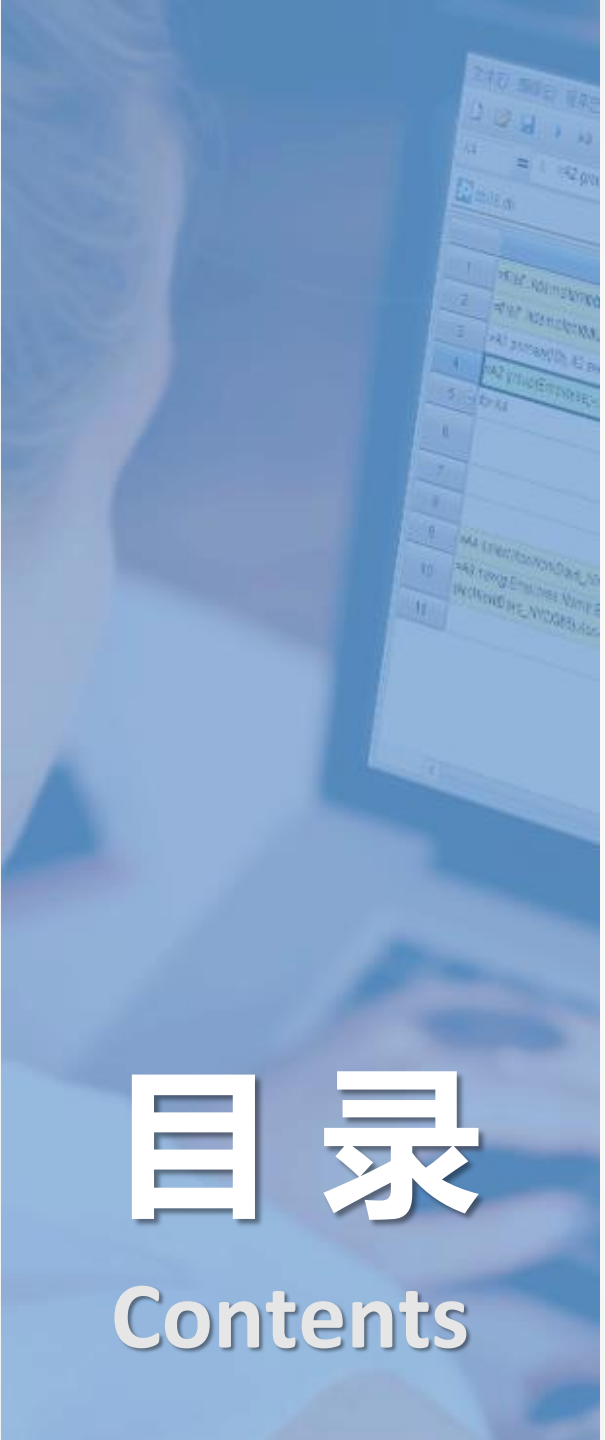
透明化提高代码兼容性，降低开发难度

透明化难以获得最优性能

高性能计算方案因场景而异，可能是矛盾的

透明化只能选择最保险的方法，一般是性能较差的那个

透明化框架对资源消耗严重



目录

Contents

集算器 – 技术特征

面向过程计算

无缝应用集成

多样性数据源接口　直接文件计算

单机优化技术　多线程并行

无中心集群结构　自由数据分布和任务分配

集算器 – 敏捷语法体系

某支股票最长连续涨了多少交易日

```
1  select max(连续日数)
2  from (select count(*) 连续日数
3        from (select sum(涨跌标志) over(order by 交易日) 不涨日数
4              from (select 交易日,
5                    case when 收盘价>lag(收盘价) over(order by 交易日)
6                    then 0 else 1 end 涨跌标志
7              from 股价表) )
8        group by 不涨日数)
```

SQL

	A
1	=股价表.sort(交易日)
2	=0
3	=A1.max(A2=if(收盘价>收盘价[-1],A2+1,0))

集算脚本

思考：按照自然思维怎么做?

语法体系更容易描述人的思路

数据模型不限制高效算法实现

集算器 – 面向过程计算

完整的循环分支控制

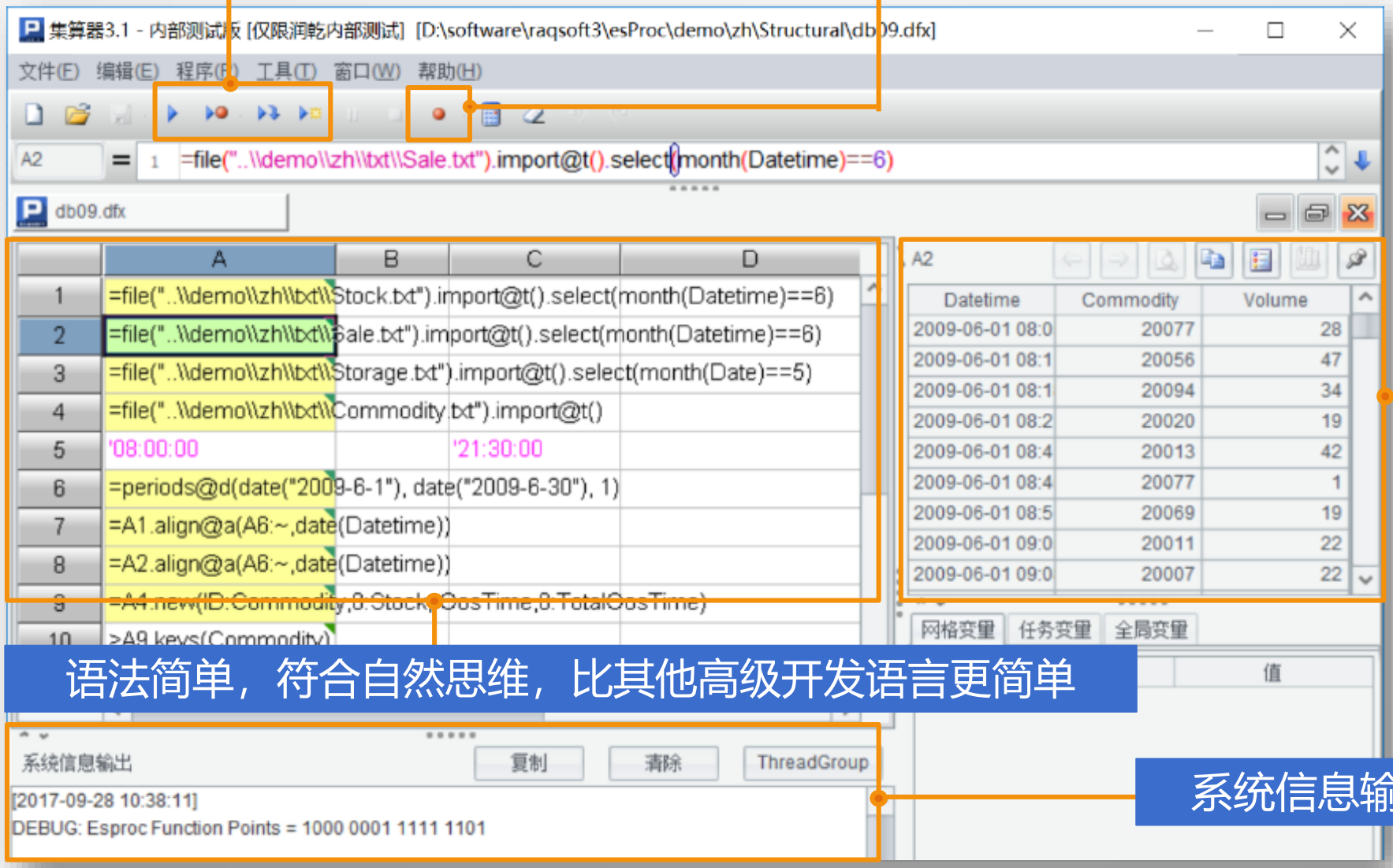
	A	B	C	D	E	F
1	=esProc.query("SELECT 订单ID AS 合同,订购日期 AS 日期,客户 ,订单金额 AS 金额,员工ID AS 销售 FROM 销售记录表 WHERE year(订购日期)=? OR year(订购日					
2	=esProc.query("select * from 员工表")					
3	>A1.run(销售=A2.select@1(编号:A1.销售))		/字段值 是记录			
4	=A1.group(销售)					
5	=create(销售,今年销售额,去年销售额,增长率,客户数,大客户数,大客户占比)					
6	for A4	=A6(1).销售.姓名				
7		=A6.select(year(日期)==年份).sum(金额)	/今年销售额			
8		=A6.select(year(日期)==年份-1).sum(金额)	/去年销售额			
9		=B8/B7-1	/增长率			
10		=A6.group(客户).(~.sum(金额))				
11		=B10.count()	/客户数			
12		=B10.count(~>A5[B9])	/大客户数			
13		=B12/B11	/大客户占比			
14		>A5.insert(0,B6,B7,B8,B9,B11,B12,B13)				
15	result A5					

天然分步、层次清晰、直接引用单元格名无需定义变量

集算器 – 开发环境

执行、调试执行、单步执行

设置断点

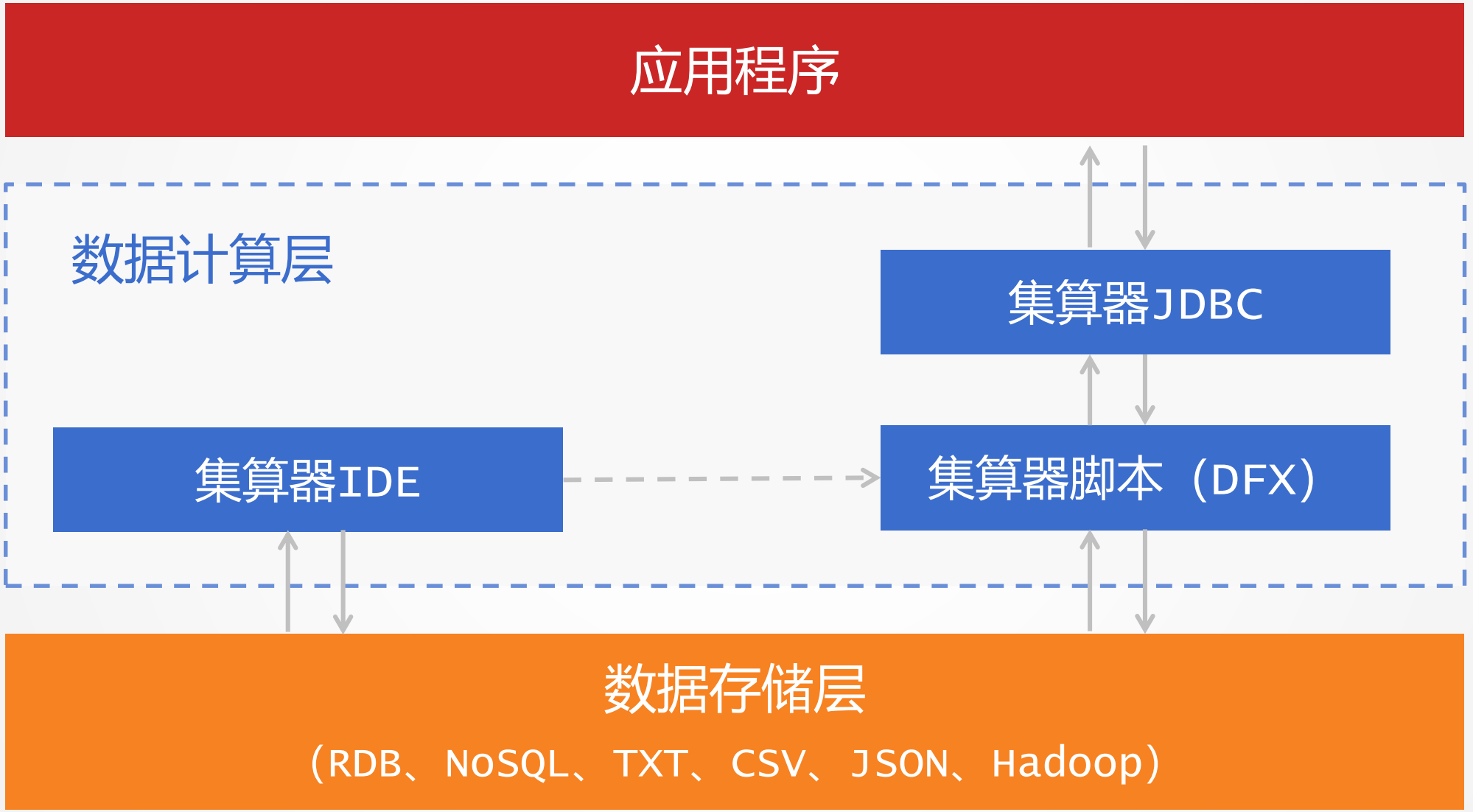


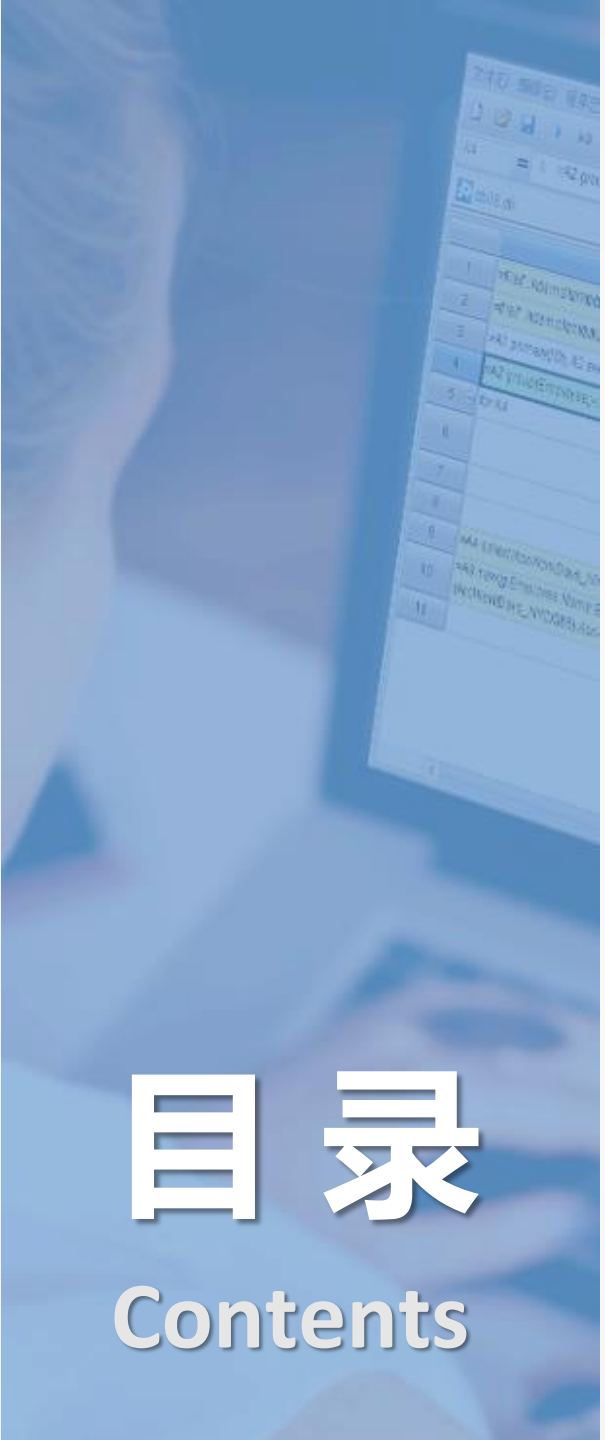
网格结果所见即所得，易于调试；方便引用中间结果

语法简单，符合自然思维，比其他高级开发语言更简单

系统信息输出，异常随时查看

集算器 – 应用结构


应用程序
数据计算层
集算器JDBC
集算器IDE
集算器脚本（DFX）
数据存储层
（RDB、NoSQL、TXT、CSV、JSON、Hadoop）

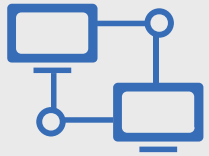


目录

Contents

高性能计算技术

遍历技术

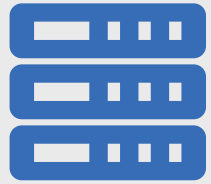


连接解决

存储格式

使用索引

分段并行



集群方案

遍历技术

遍历是大数据计算的基础

遍历技术 延迟游标

游标概念

流式读入数据，每次仅计算一小部分

延迟计算

在游标上定义运算，返回结果仍然是游标，可再定义运算

不立即计算，最终一次性遍历和计算

	A	B
1	=file(“data.txt”).cursor@t()	/创建游标
2	=A1.select(product==“1”)	/过滤
3	=A2.derive(quantity*price:amount)	/计算列
4	=A3.sum(amount)	/实际计算

遍历技术 遍历复用

外存计算优化方向是减少访问量

可复用的遍历减少外存访问量

	A	
1	=file(“data.txt”).cursor()	
2	=channel().groups(;count(1))	配置同步计算
3	>A1.push(A2)	绑定
4	=A1.sortx(key)	排序，遍历过程中处理绑定计算
5	=A2.result().#1	取出绑定计算的结果，即总记录数
6	=A4.skip((A5-1)\2).fetch@x(2-A5%2).avg(key)	取出中位数记录并计算中位数

一次遍历可返回多个分组结果

遍历技术 聚合理解

从一个集合计算出一个单值或另一个集合都可理解为聚合

高复杂度的排序问题转换为低复杂度的遍历问题

	A	
1	=file(“data.txt”).cursor@t()	
2	=A1.groups(;top(10,amount))	金额在前10名的订单
3	=A1.groups(area;top(10,amount))	每个地区金额在前10名的订单

遍历技术 有序游标

针对已有序的数据可一次遍历实现大结果集分组运算，减少外存交换

	A	
1	=file(“data.txt”).cursor@t()	
2	=A1.groupx@o(uid;count(1),max(login))	

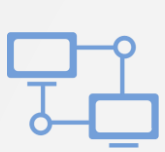
复杂处理需要读出到程序内存中再处理

有序游标有效减少查找和遍历数量

	A	B	C
1	=file(“user.dat”).cursor@b()		/按用户id排序的源文件
2	for A1;id	…	/从游标中循环读入数据，每次读出一组id相同
3		…	/处理计算该组数据

连接解决

连接计算是结构化数据的最大难点



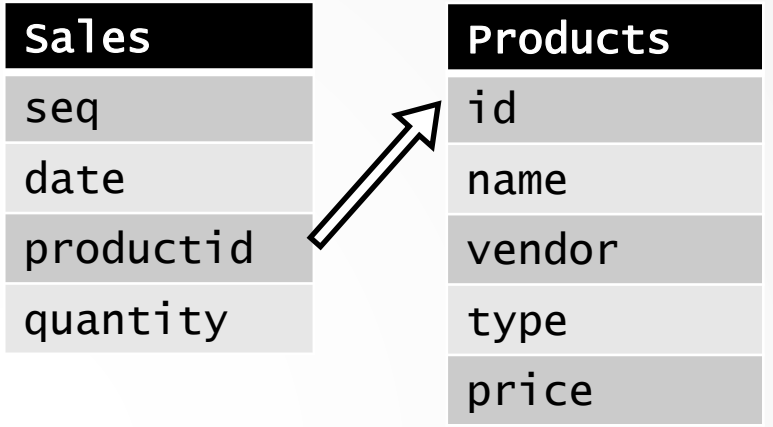
连接解决 区分JOIN!

外键维表1:N	同维表1:1	主子表1:N
指针化 序号化	有序归并	有序归并

连接解决 外键指针化

外键需要随机小量频繁访问

内存指针查找大幅提高性能



	Java指针连接	Oracle
单表无连接	0.57s	0.623s
五表外键连接	2.3s	5.1s

	A	
1	=file(“Products.txt”).import()	读入商品列表
2	=file(“Sales.txt”).import()	读入销售记录
3	>A2.switch(productid,A1:id)	建立指针式连接，把商品编号转换成指针
4	=A2.sum(quantity*productid.price)	计算销售金额，用指针方式引用商品单价

连接解决 外键序号化

序号化相当于外存指针化

	A	
1	=file(“Products.txt”).import()	读入商品列表
2	=file(“Sales.txt”).cursor()	根据已序号化的销售记录建立游标
3	=A2.switch(productid,A1:#)	用序号定位建立连接指针，准备遍历
4	=A3.groups(;sum(quantity*productid.price))	计算结果

不需要再计算Hash值和比较

连接解决 有序归并

同维表和主子表连接可以先排序后变成有序归并

追加数据的再排序也仍然是低成本的归并计算

	A	
1	=file(“Order.txt”).cursor@t()	订单游标，按订单id排序
2	=file(“Detail.txt”).cursor@t()	订单明细游标，也按订单id排序
3	=joinx(A1:O,id;A2:D,id)	有序归并连接，仍返回游标
4	=A3.groups(O.area;sum(D.amount))	按地区分组汇总金额，地区字段在主表中，金额字段在明细子表中

存储格式

有效的存储格式是
高性能的保证

存储格式 压缩二进制

数据类型已存入，无须解析

轻量级压缩

减少硬盘空间
很少占用CPU时间

泛型存储，允许集合数据

可追加

存储格式 自由列存

自由分配列组

行列存统一

重复值压缩

对上透明访问

组1	组2		组k	
列1	列2	列3	列4	…
1	…	…	…	
2	…	…	…	
…				
1023				
1024				

过滤优化

只读取与条件相关的列组

存储格式 序号主键

多层序号式主键

外存指针式外键，高速引用

外存游离记录表示，离散性

天然有序，易于查找

分组针对外键

结果自然对齐有序

北京		1
	海淀	1,1
	朝阳	1,2
	…	
上海		2
	浦东	2,1
	…	
…		

存储格式 主子合一

多层复式表

层次式有序集合

每层均可以有数据结构

同维表与主子表统一

消除对齐式连接

订单		客户	地区	日期	...
	订单明细	产品	数量	单价	...

使用索引

有序对分查找

有序数据提供对分查找，快速定位

普通定位索引

按键值找到数据

两段式索引提高维护性能

片状索引

连续记录的索引

使用索引 应用：切片的双向逆序索引

CUBE切片索引的困难

列存索引太大

数据必须实际排序

双向逆序索引

按D1,…,Dn和Dn,…,D1双倍排序

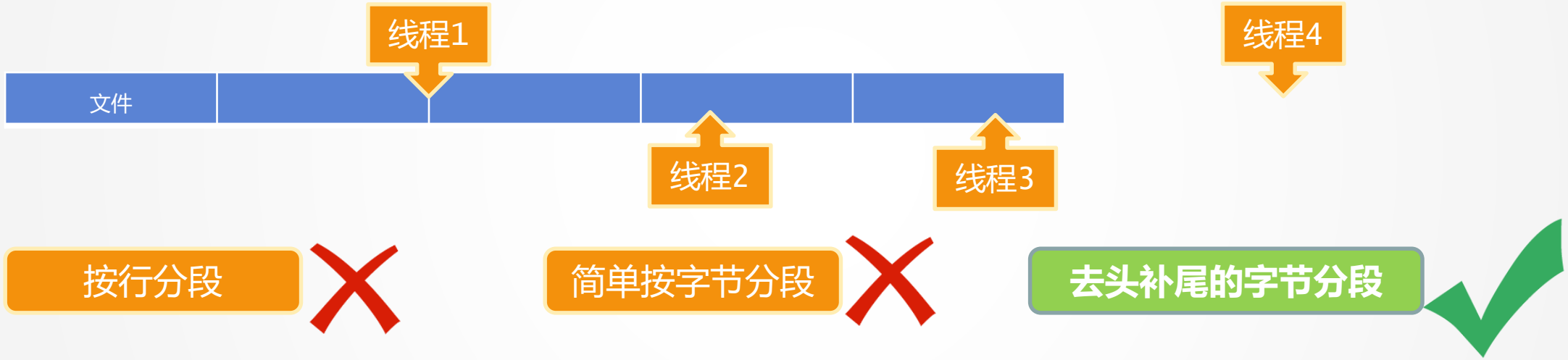
只针对前半部分维度使用片式索引

分段并行

数据分段是并行计算的基础

分段并行 文本分段

并行处理需要将数据文件分段，每个线程处理一段



文本并行解析

	A	
1	=file(“data.txt”).cursor@tm(amount)	/定义并行取数的游标（并行）
2	=A1.groups(;sum(amount):amount)	/遍历游标汇总amount（串行）

分段并行 倍增分段

分段数足够大，以适应变化的并行数

每段记录数相对平均

数据追加时不必重写所有数据，保持连续性

	A	B	
1	=file(“data.bin”)		
2	fork 4	=A1.cursor@b(amount;A2:4)	
3		=B2.groups(;sum(amount):a)	
4	=A2.conj().sum(a)		

追加前	追加后
1	1
2	
3	2
4	
5	3
6	
7	4
8	
…	
1023	512
1024	
	513
	…

分段并行 列存分段

倍增分段方案解决列存并行困难

各列同步分段

同列连续存储

	A	B	
1	=[1,3].(file("col"/~/".bin"))		
2	fork 4	=A1.cursor(amount;A2:4)	
3		=B2.groups(;sum(amount):a)	
4	=A2.conj().sum(a)		

段	列1	列2	…
1	1,…,k	1,…,k	
2	k+1,…,2k	k+1,…,2k	
3	2k+1,…,3k	2k+1,…,3k	
4	3k+1,…,4k	3k+1,…,4k	
…			
1023			
1024			

分段并行 有序与对位分段

有序数据的分段点要落在组边界上才能分段并行

	A	
1	=file(“userlog.txt”).cursor@t()	原始数据游标
2	=A1.sortx(id)	按id排序
3	>file(“userlog.dat”).export@z(A2;id)	写成按id分段的文件

同维表、主子表需同步分段

	A	
1	=file(“Order.txt”).cursor@t()	
2	=A1.sortx(id)	按id排序
3	>file(“Order.dat”).export@z(A2;id)	写成按id分段的文件
4	=file(“Detail.txt”).cursor@t()	
5	=A4.sortx(id)	按id排序
6	>file(“Detail.dat”).export@z(A2;id,file(“Order.dat”),id)	和Order.dat同步按id分段

分段并行 多路游标

建立多路游标，可继续进行过程计算，会自动并行执行，简化书写难度。

	A	
1	=file(“Order.txt”).cursor@tm()	建立多路游标，自动并行执行
2	=A1.select(month(Date)==10)	条件过滤
3	=A2.groups(ID;sum(COST*WEIGHT):VALUE)	分组汇总

集群方案

集群设计原则

确保容错

减少网络传输量

考虑负载均衡

集群计算技术

集群方案 – 无中心设计

服务器集群无中心

集算器没有框架，没有永久的中心主控节点，程序员用代码控制参与计算的节点

无中心不会发生单点失效

所有节点地位都等同，不会发生单点失效，某个节点有故障时整个集群仍然可以运行

计算任务有主控节点

在计算过程中由主控节点临时寻找其它节点参与计算

集群方案 – 负载均衡与容错

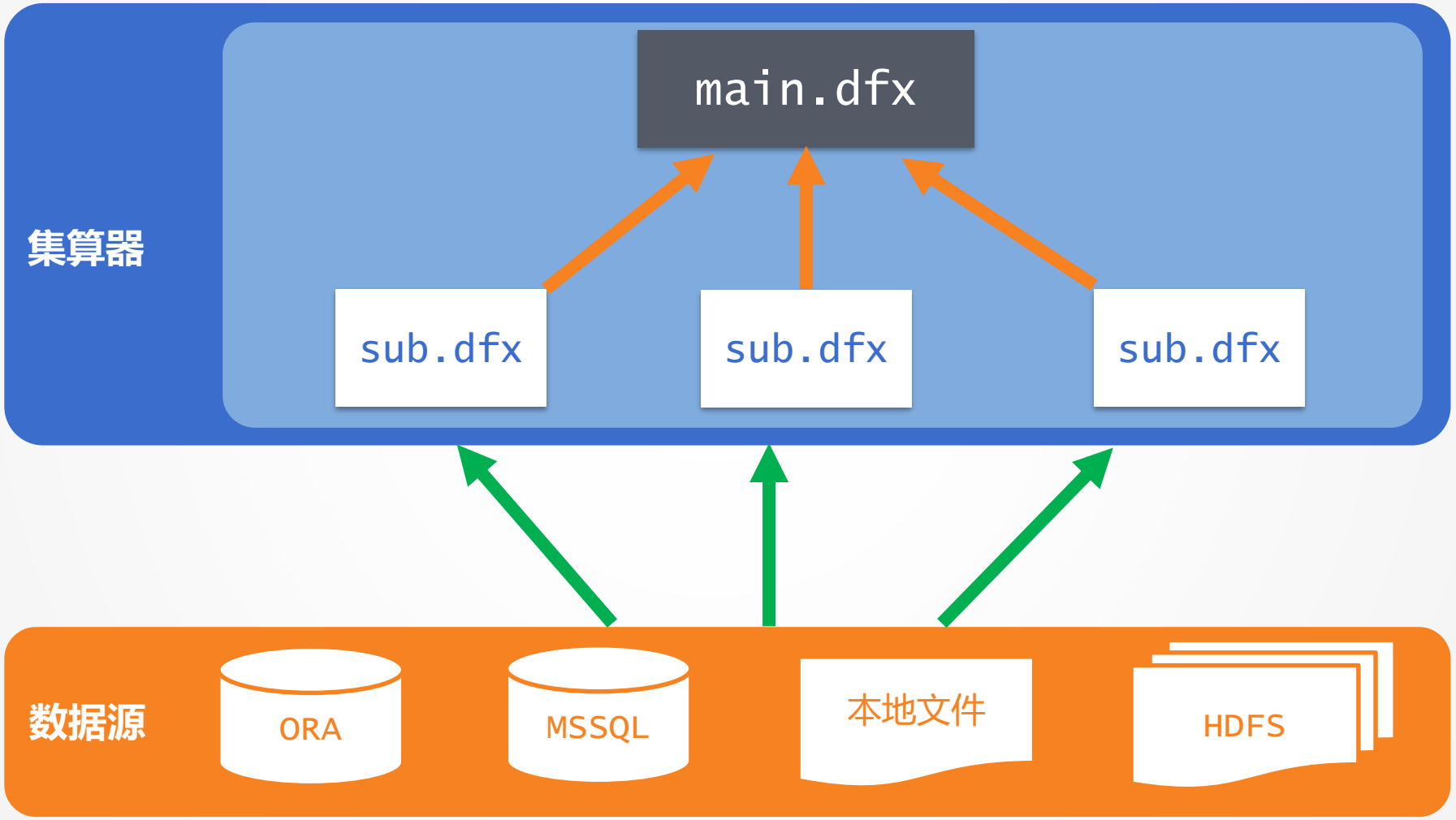
具备负载均衡能力

根据每个节点空闲程度（当前正在运行的线程数量）决定是否分配任务，实现负担和资源的有效平衡

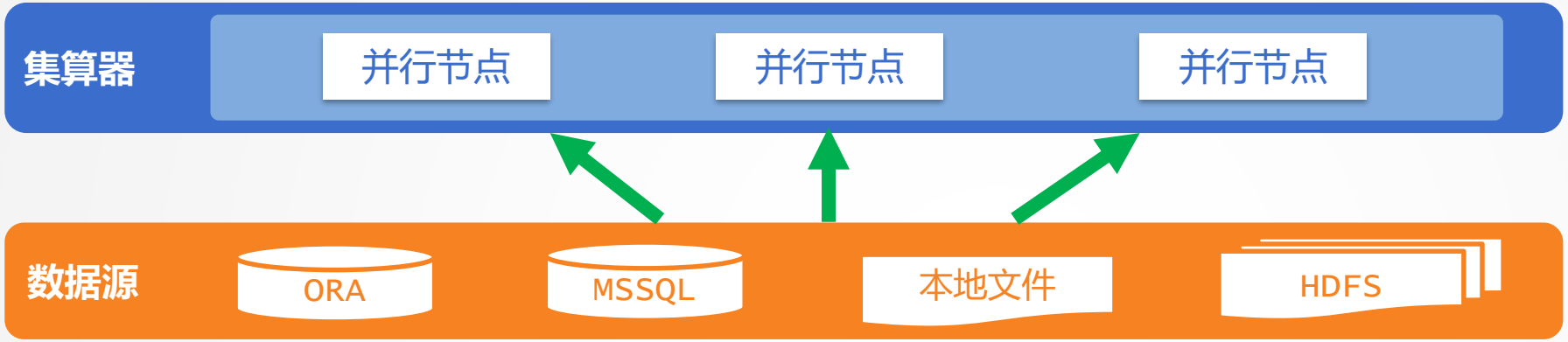
具备容错能力

某个节点失效导致子任务失败，主控程序还会再次将这个子任务分配给其它有效的节点

集群方案 – 并行逻辑结构



共享式数据计算分布



共享数据源方式：计算分布实现，数据共享读取

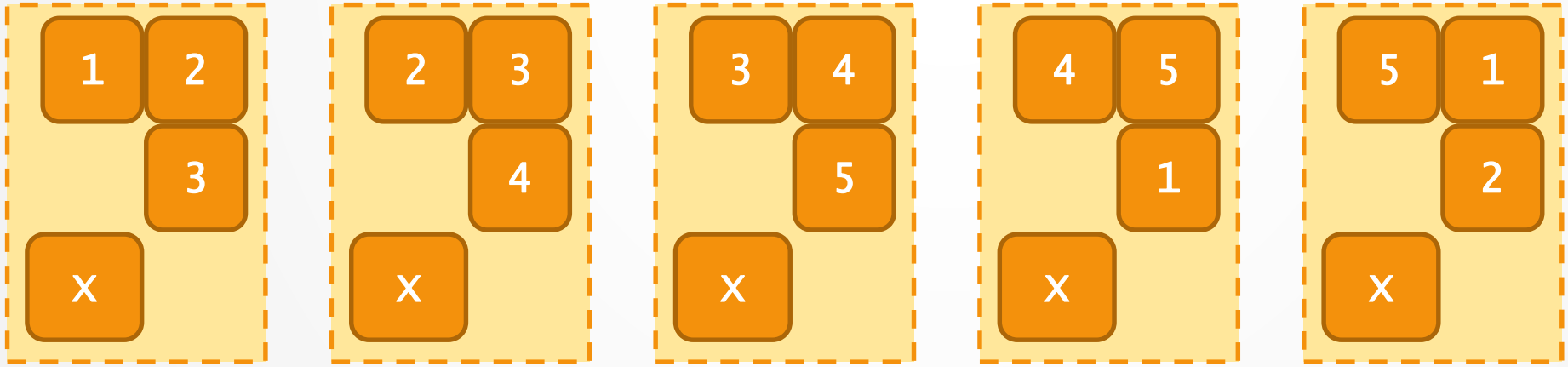
	A	B	
1	=4.(“192.168.0.”/(10+~)/”:1234”)		节点机列表，4个
2	fork to(8);A1		到节点机上执行，分成8个任务
3		=hdfsfile(“hdfs:\\192.168.0.1\persons.txt”)	HDFS上的文件
4		=B3.cursor@t(;A2:8)	分段游标
5		=B4.select(gender==’M’).groups(;count(1):C)	过滤并计数
6	=A2.conj().sum(C)		汇总结果

冗余式外存分布

所有任务都需要用到的公共维表复制存储

事务数据分成N个区，根据需要的容错指数循环分区

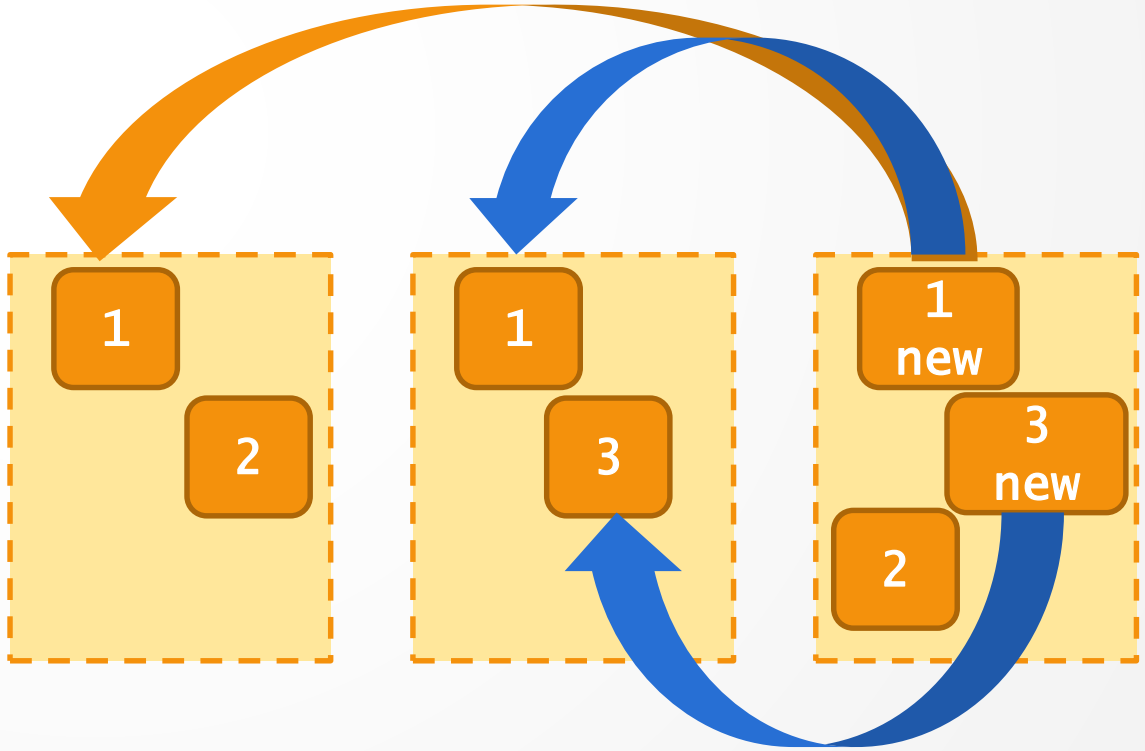
存储利用率约为1/k（允许k-1个节点失效）



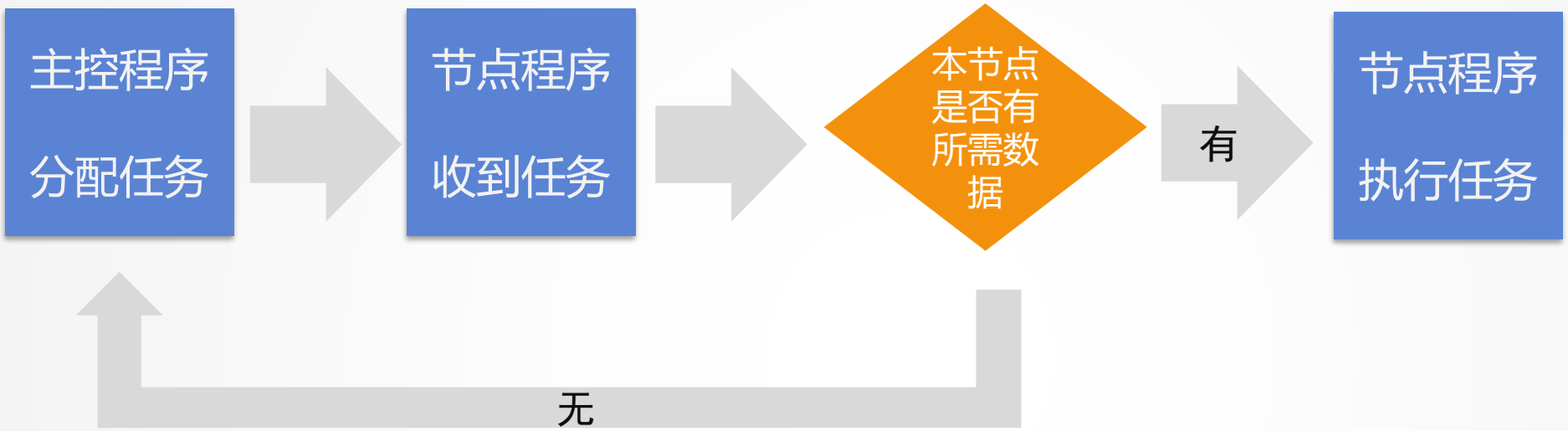
外存分布 – 数据同步

集算器提供节点之间的同区数据同步功能

每个节点都是独立的计算机，可以应用内存和外存的计算方法。



外存分布 – 动态任务分配



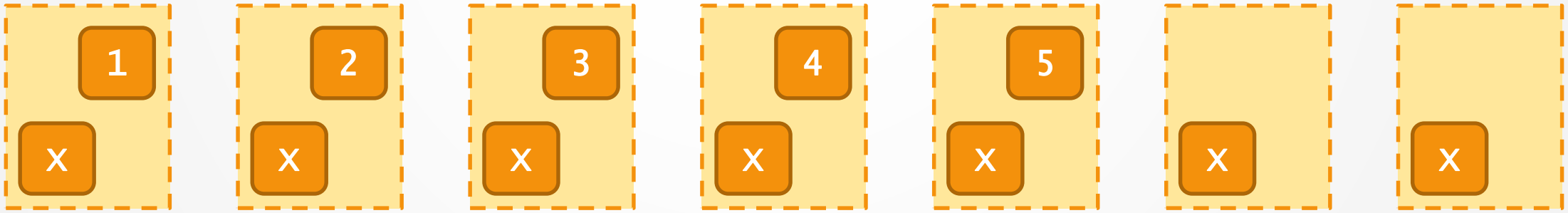
	A	B	C	
1	=4.(“192.168.0.”/(10+~)/”:1234”)			节点机列表，4个
2	fork to(8);A1			到节点机上执行，分成8个任务
3		=file(“person.txt”,A2)		A2为数据区号
4		if !B3.exists()	end “data not find”	找不到返回错误再分配
5		=B3.cursor()		
6		=B5. select(gender==’M’).groups(;count(1):C)		计算
7	=A2.conj().sum(C)			汇总

备胎式内存分布

数据分区分别加载进节点内存

n个节点（每节点一个分区）+k个备份节点

内存利用率n/(n+k)



内存分布 – 静态任务分配

预留备份节点（备胎）不加载任何数据分区

发现有分区在所有可用节点都找不到时，启动找备份节点执行加载该分区

任务直接分配到相应节点，不再动态询问

	A	主程序
1	=8.("192.168.0."/(10+~)/":1234")	节点列表，共8个
2	=hosts(A1,to(4),"init.dfx")	在其中寻找4个节点加载内存数据
3	=callx@a("sub.dfx",to(4);A2)	调用这些节点上程序计算
4	=A3.sum()	汇总结果

	A	节点初始化程序init.dfx
1	=file("Product.txt",z).import()	读入第z分区的数据
2	>env(T,A1)	将数据记入环境
3	>zone(z)	登记本节点的分区

	A	节点程序sub.dfx
1	=T	从环境中取出数据
2	=A1.count(gender=='M')	过滤并计数
3	return A2	返回结果

MapReduce->ForkReduce

MapReduce的问题

任务太碎，管理成本过高

难以实现关联运算

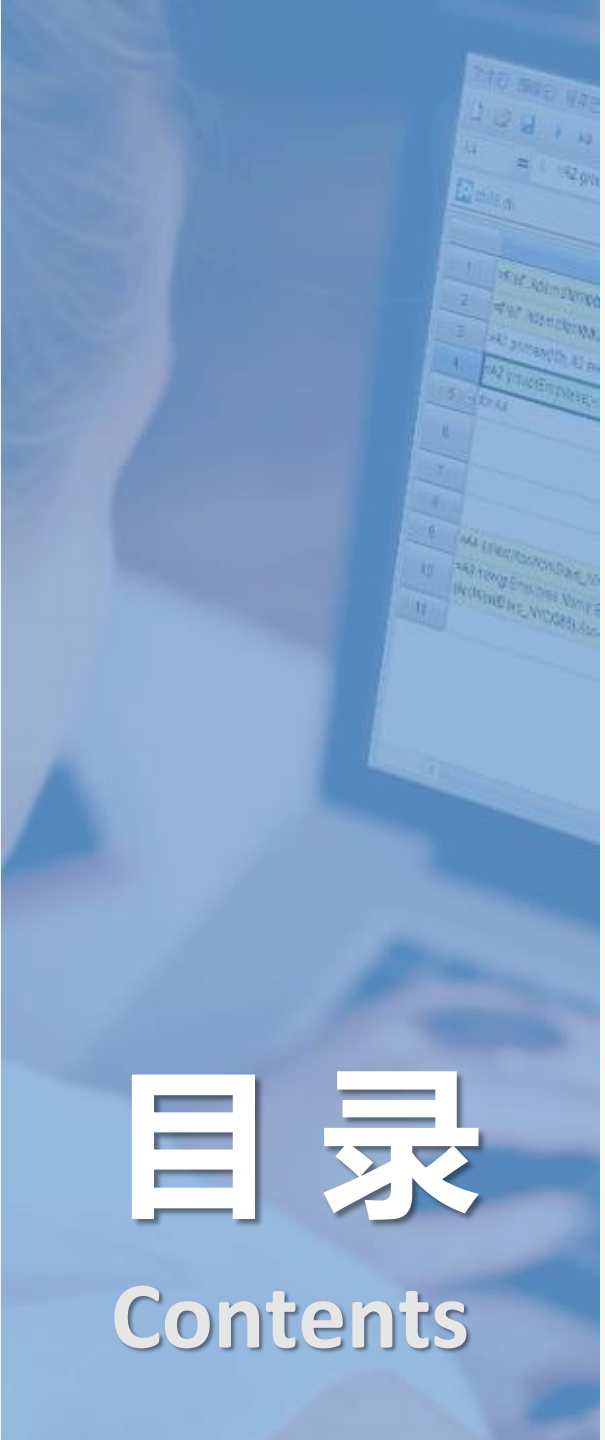
Hash Shuffle随意性太强

ForkReduce

批量任务，调度成本低

结合对位分段技术实现关联运算

Shuffle结果有确定分布方案



目录

Contents

某大型保险公司 - 存储过程优化

【案例背景】定报价风险保费通过informix存储过程计算，运行1小时，效率低下影响使用；初始化车险新规车辆归并,找到上三年保单，数据量大（30天）时，运行将近2小时，急需优化

【数据规模】保单表：0.35亿；保单明细表：1.23亿

【优化效果】

场景	优化前	优化后	提升倍数
定报价风险保费	3600秒	68秒	52.9倍
查找上三年保单	6672秒	1020秒	6.5倍

52.9倍

【使用的集算器技术】

压缩列存：使用集算器组表的列式压缩存储，空间小，并且减少了参与计算的列数（约1/7）

有序存储：数据按照保险单号顺序存放，这样可以使用有序关联提升计算效率

高效计算：原来需要遍历多次完成多表关联，现在只需遍历一次即可完成多表关联

并行技术：集算器提供多线程并行计算，在多核环境下可以加速计算

某金融机构 – 报表优化

【案例背景】该金融机构需要维护的报表超过7500张，其中20%查询速度为分钟级，5%的查询速度为小时级，这5%非常慢的查询SQL代码行数有几百行，存储过程更是达到几千行，关联运算多。用通俗的语言描述报表慢的原因主要是数据量大，业务逻辑复杂。

【数据规模】POS交易情况统计表运行时间为3700秒，4个SQL数据集，涉及6个数据库表的关联运算，其中3个表的数据量为千万级（1708万-4886万）。

【优化效果】不改变硬件环境的条件下，报表查询速度从3700秒降到105秒。

【使用的集算器技术】

高效关联技术：将原来报表的顺序关联改为更高效的外键指针式关联方式

并行计算：将原有串行计算的4个SQL，改为4线程并行加速计算

35.2倍

某银行 – 报表优化

【案例背景】随着客户业务数据量级的日益增长，使原有业务报表在查询和导出时的效率明显降低，已经跟不上业务的发展；并且在引入HBase后，仍无法解决批量查询效率问题。

【数据规模】千万级

【优化效果】原有回单报表查询一个分行一个月的数据需要18分钟，甚至造成内存溢出，引入集算器后，每页展示2000条数据，首页加载时间为10秒，一个分行一个月的数据全部加载完成缩短至5分钟，并且不会造成内存溢出的问题。

【使用的集算器技术】

高性能存储：集算器提供了私有二进制压缩存储，提供保存数据类型、泛型存储、任意分段等机制

双线程异步计算：与报表工具结合，将取数计算和数据呈现使用不同的异步线程处理，提升数据展现速度，并降低内存使用率

3.6倍

某电信运营商 – Excel财务报表优化

【案例背景】月财务结算时需要各省市分级按科目上报财务报表，但由于集团下属分子公司使用了不同的财务软件（SAP和用友），上报的数据存在财务账套无法完全匹配，不同科目需要关联替换等业务数据准备工作需要财务人员手工完成，使用Excel进行报表统计十分低效。

【数据规模】千万级

【优化效果】以费用/利润统计表为例，查询时间由原来317秒，优化到13秒

【使用的集算器技术】

高性能存储：集算器提供了私有二进制压缩存储，提供保存数据类型、泛型存储、任意分段等机制

热切换：集算器SPL采用解释执行机制，报表变更时直接修改SPL实施热部署

24.4倍

某省联通– ETL优化

【案例背景】用户的ETL主要通过行业内DataStage进行，普遍存在跑数时间过长，长时间的连接访问数据源库，一方面影响数据生产，另一方面对源库直接占用资源

【数据规模】3.5亿

【优化效果】以开发订购类指标（活动日表）的2/3G部分计算为例，原DS处理需要3.5小时，使用集算器优化后运行1.37小时

【使用的集算器技术】

高性能存储：集算器提供了私有二进制压缩存储，提供保存数据类型、泛型存储、任意分段等机制

库外计算：集算器作为独立计算引擎可以实施不依赖于数据库的库外计算，避免对数据库造成影响

并行计算：集算器采用并行计算加速ETL过程

2.6倍

某大型零售企业 – 库龄计算

【案例背景】库龄计算是零售行业的常见场景，以往采用Oracle存储过程计算效率低下，长时间占用数据库资源得不到释放，影响其他业务

【数据规模】3亿

【优化效果】9分钟优化到10秒

【使用的集算器技术】

高性能存储：集算器提供了私有二进制压缩存储，提供保存数据类型、泛型存储、任意分段等机制

外存游标技术：支持数据量大时采用外存游标分批加载到内存进行计算

并行计算：集算器可实施单机多线程和多机分布式并行计算

54倍

其他案例

客户	场景	效果	提升倍数	使用集算器技术
某部委	报表加速	查询速度：330秒→10秒 导出速度：1050秒→100秒	查询：**33**倍 导出：**10.5**倍	集算器高效关联技术
某运营商	指标计算	优化前：3.9小时 优化后：1.44小时	**2.7**倍	高效存储+并行计算
某省邮政	报表优化	优化前：900秒 优化后：32秒	**28**倍	高性能存储+有序计算+索引技术
某省财政厅	指标查询优化	优化前：270秒 优化后：30秒	**9**倍	高效存储+异步计算+外存游标

创新技术
推动应用进步！


各设备故障分析
近7天设备的总故障率
近7天
近1天
近6小时
1.53%
0.95%
2.40%
近7天故障情况
全年故障综合分析
近7天故障次数前5的设备
服务器CPU使用率
今日
昨日
今天硬盘情况