

第十一章 流计算

1. 核心概念与背景

1.1 静态数据 vs 流数据

- **静态数据 (Static Data):**
 - **特点：**大量历史数据，存储在数据仓库中，是过去某一时刻的快照。
 - **处理模式：**批量计算 (Batch Processing)，如 Hadoop MapReduce。
 - **交互：**用户主动发起查询 (Pull)。
- **流数据 (Stream Data):**
 - **特点：**
 - 数据快速持续到达，潜在大小也许是无穷无尽的；
 - 数据来源众多，格式复杂；
 - 数据量大，但是不十分关注存储，一旦经过处理，要么被丢弃，要么被归档存储；
 - 注重数据的整体价值，不过分关注个别数据；
 - 数据顺序颠倒，或者不完整，系统无法控制将要处理的新到达的数据元素的顺序
 - **处理模式：**实时计算/流计算。
 - **交互：**系统主动推送结果 (Push)。

1.2 批量计算和实时计算

- 对于静态数据和流数据的处理对应着两种截然不同的计算方式：批量计算和实时计算
- **批量计算：**

充裕时间处理静态数据，如Hadoop。
流数据不适合采用批量计算，因为流数据不适合用传统的关系模型建模。
流数据必须采用实时计算，响应时间为秒级
数据量少时，实时处理都不是问题，但是，在大数据时代，数据格式复杂、来源众多、数据量巨大，对实时计算提出了很大的挑战。因此，针对流数据的实时计算——流计算，应运而生
- **流计算：**

流计算是实时获取来自不同数据源的海量数据，经过实时分析处理，获得有价值的信息

1.3 流计算的需求

流数据的价值随着时间流逝而降低，因此对于流计算系统来说，**流数据应达到如下需求：**

- **高性能：**处理大数据的基本要求，如每秒处理几十万条数据；
- **海量式：**支持TB级甚至是PB级的数据规模；
- **实时性：**保证较低的延迟时间，达到秒级别，甚至是毫秒级别；

- 分布式：支持大数据的基本架构，必须能够平滑扩展；
- 易用性：能够快速进行开发和部署；
- 可靠性：能可靠地处理流数据

1.4 为什么要流计算？（Hadoop的局限性）

- Hadoop设计的初衷是面向大规模数据的批量处理，每台机器并行运行MapReduce任务，最后对结果进行汇总输出。
 - MapReduce是专门面向静态数据的批量处理的，内部各种实现机制都为批处理做了高度优化，不适合用于处理持续到达的动态数据。即使将 MapReduce 切分成小片段（Micro-batch），也无法满足毫秒级低延迟需求。
 - **结论：**Hadoop 擅长批处理，不适合流计算。
-

2 流计算处理流程

2.1 典型的流计算包含三个阶段：

1. **数据实时采集：**
 - **挑战：**数据分散、量大。
 - **工具：**Kafka, Flume, Scribe, Time Tunnel。
 - **架构：**Agent (采集) -> Collector (转发/汇聚) -> Store (暂存/队列)。
2. **数据实时计算：**
 - 对采集的数据进行实时分析。
 - 数据流向：流入 -> 系统处理 -> 流出/丢弃/存储。
3. **实时查询服务：**
 - 将结果主动推送给用户，或更新仪表盘。
 - **区别：**传统查询是针对“过去”的数据，实时查询是针对“当下”不断更新的数据。

2.2 流处理系统与传统的数据处理系统的不同

- 流处理系统处理的是实时的数据，而传统的数据处理系统处理的是预先存储好的静态数据。
 - 用户通过流处理系统获取的是实时结果，而通过传统的数据处理系统，获取的是过去某一时刻的结果。
 - 流处理系统无需用户主动发出查询，实时查询服务可以主动将实时结果推送给用户。
-

3. 核心框架一：Storm

3.1 简介

- **定位：**免费、开源的分布式实时计算系统（地位类似于批处理中的 Hadoop）。
- **特点：**
 - 整合性：Storm可方便地与队列系统和数据库系统进行整合。
 - 简易的API：Storm的API在使用上即简单又方便。
 - 可扩展性：Storm的并行特性使其可以运行在分布式集群中。
 - 容错性：Storm可自动进行故障节点的重启、任务的重新分配。
 - 可靠的消息处理：Storm保证每个消息都能完整处理。
 - 支持各种编程语言：Storm支持使用各种编程语言来定义任务。
 - 快速部署：Storm可以快速进行部署和使用。
 - 免费、开源：Storm是一款开源框架，可以免费使用。

3.2 核心概念 (设计思想)

- **Streams (流)：**无界的 Tuple 序列。
- **Tuple (元组)：**数据传输的基本单元（值列表）。
- **Spout (喷口)：**
 - Stream 的源头。
 - 负责从外部数据源（如 Kafka）读取数据并封装成 Tuple 发送。
- **Bolt (螺栓)：**
 - 逻辑处理单元。
 - 执行过滤、聚合、Join、写库等操作。
 - 可以发射新的 Stream 给下一个 Bolt。
- **Topology (拓扑)：**
 - 由 Spout 和 Bolt 组成的有向图（网络）。
 - 对比：MapReduce Job 最终会结束，Topology 会一直运行直到被手动终止。
- **Stream Groupings (分组策略)：**决定 Tuple 如何分发给下一个 Bolt 的 Task。
 - ShuffleGrouping：随机分组，随机分发Stream中的Tuple，保证每个Bolt的Task接收 Tuple数量大致一致。
 - FieldsGrouping：按照字段分组，保证相同字段的Tuple分配到同一个Task中。
 - AllGrouping：广播发送，每一个Task都会收到所有的Tuple。
 - GlobalGrouping：全局分组，所有的Tuple都发送到同一个Task中。
 - NonGrouping：不分组，和ShuffleGrouping类似，当前Task的执行会和它的被订阅者在同一个线程中执行。
 - DirectGrouping：直接分组，直接指定由某个Task来执行Tuple的处理。

3.3 集群架构 (Master-Worker)

- Master节点运行名为“Nimbus”的后台程序（类似Hadoop中的“JobTracker”），负责在集群范围内分发代码、为Worker分配任务和监测故障。
- Worker节点运行名为“Supervisor”的后台程序，负责监听分配给它所在机器的工作，即根据Nimbus分配的任务来决定启动或停止Worker进程，一个Worker节点上同时运行若干个

Worker进程。

3.4 并行度概念 (层级)

1. **Worker (进程)**: 运行在 Supervisor 节点上, 属于某个 Topology。
2. **Executor (线程)**: 运行在 Worker 内部, 执行具体的 Task。
3. **Task (实例)**: 实际的数据处理逻辑单元。
 - 关系: 默认情况下 Executor 数量 = Task 数量 (一个线程执行一个任务)。

3.5 storm工作流程

- 所有Topology任务的提交必须在Storm 客户端节点上进行, 提交后, 由Nimbus 节点分配给其他Supervisor节点进行处理。
 - Nimbus节点首先将提交的Topology进 行分片, 分成一个个Task, 分配给相应的 Supervisor, 并将Task和Supervisor相关的信息提交到Zookeeper集群上。
 - Supervisor会去Zookeeper集群上认领 自己的Task, 通知自己的Worker进程进 行Task的处理。
-

4. 核心框架二：Spark Streaming

4.1 基本原理

- **微批处理 (Micro-batch)**: 将实时流拆分为很小的时间片 (如 1秒), 然后由 Spark 引擎按“批处理”的方式处理。
- **核心抽象: DStream (Discretized Stream)**
 - 代表连续的数据流。
 - 内部实现为一系列连续的 RDD。
 - 对 DStream 的操作最终转变为对底层 RDD 的操作。

4.2 特点

- 支持多种数据源 (Kafka, Flume, HDFS, TCP socket)。
 - 基于 Spark 生态, 易于与 Spark SQL, MLlib, GraphX 整合。
-

5. 重点对比: Storm vs Spark Streaming

这是考试和选型中最关键的部分:

特性	Storm	Spark Streaming
实时性/延迟	毫秒级 (纯实时)	秒级/百毫秒级 (微批处理)
处理模型	以此 Tuple 为单位处理	以时间片(Batch)为单位处理 RDD
编程灵活性	通过 Thrift 支持多语言	一体化编程 (SQL/ML/Graph)
容错性	较好 (Ack机制)	更好 (基于 RDD 血统)
适用场景	极度追求低延迟 (如高频交易)	需要复杂分析、历史与实时结合 (如推荐系统)

6. 应用场景举例

1. **实时分析**: 淘宝“双11”大屏，实时统计交易额（避免离线计算的滞后）。
2. **实时推荐**: 根据用户实时浏览轨迹推荐内容。
3. **实时交通**: 基于当前车流进行导航规划。
4. **网络监控**: 实时检测 DDoS 攻击。