

实时流计算在航天地面数据处理系统中的应用

李一辰^{1,2}, 李绪志¹, 阎 镇¹

(1 中国科学院 空间应用工程与技术中心, 北京 100094; 2 中国科学院大学, 北京 100190)

摘 要: 针对航天地面系统中数据高速、实时和突发性等特点, 以实时流计算 Storm 系统解析 AOS 在轨协议数据为例, 讨论了实时流计算系统在航天系统中的应用和优势, 同时指出现有实时流计算框架不适合处理连续型数据流的不足, 并给出通过并行处理提高处理速度的解决方案及性能评价。

关键词: 实时流计算; Storm; AOS 在轨协议; 并行处理

中图分类号: TP391

文献标识码: A

文章编号: 1000-7180(2014)09-0015-05

Real-time Stream Computing in Aerospace Ground System's Data Disposing

LI Yi-chen^{1,2}, LI Xu-zhi¹, YAN Zhen¹

(1 Technology and Engineering Center for Space Utilization, Chinese Academy of Sciences, Beijing 100094, China;

2 Graduate University of Chinese Academy of Sciences, Beijing 100190, China)

Abstract: Real-time stream computing system is very fit to deal with online and outburst high speed data in aerospace systems. Using Storm and AOS protocol data as an example, explain the superiority for making up with real-time stream computing system. For Storm is not suitable to resolve time referenced data, put forward the parallel computing to increase the processing speed and give the performance evaluation.

Key words: real-time computing; storm; AOS; parallel computing

1 引言

随着航天技术的发展, 飞行器上配置了越来越多的载荷, 同时中继技术的采用使得卫星能够更频繁地上传包括图像、科学数据、工程数据等在内的多种信息, 地面系统中的交互、查询等请求也越来越频繁, 这些都需要地面系统能够实时地支持高速数据处理。如何解决上述问题, 是现在航天系统所面临的共同问题^[1]。

目前在互联网领域中处理大数据时通常采用 Hadoop^[2] 批处理方式, 而航天领域中 SkyBox^[2] 公司用 Hadoop 进行卫星图片处理获得了非常好的效果。但是由于批处理方式会带来一定延迟, 并不适合航天领域中实时处理的需求。为提高系统响应时间和降低处理延迟, 众多公司采用实时流计算系统来

处理高速突发数据流, 如 Twitter 的 Storm、Yahoo! 的 S4 等^[3], 在处理高速事务单元方面取得了非常好的效果^[3]。本文将介绍如何利用现有的实时流计算系统来处理航天领域中的下行数据, 并以 AOS^[4] 数据处理为例说明实时流计算系统的应用, 同时对现有实时流计算系统不适用于航天数据处理的地方提出改进, 并给出设计和实现方案。

2 实时流计算技术简介

实时流计算系统是一个概念式的分布式并行计算系统, 称之为概念式是因为目前实时流计算并没有统一的框架可以遵循, 各个公司和单位都可以根据自己需要对现有多个开源框架进行扩展或者自行搭建。本文将会以目前使用最多的开源实时流系统 Storm 为例, 说明实时流计算系统的框架结构和编程方式^[5]。

2.1 实时流计算框架

实时流计算主要应用于两个场景:持续计算,一旦任务启动除非人为终止否则会一直处理到达数据;支持分布式并行计算,对大数据采用并行计算以降低单个处理器任务量^[3]. 实时流计算系统处理模型如图 1 所示。

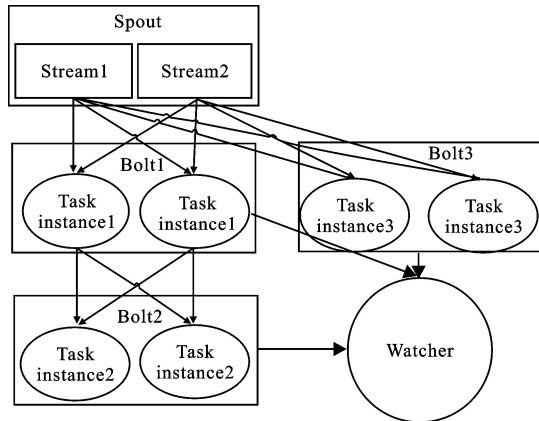


图 1 实时流计算系统的处理模型

处理模型中各模块定义如下:

(1)数据源节点(Spout),数据消息产生者,能够提供可靠消息和不可靠消息;

(2)数据流(Stream),在时间分布和数量上无限的一系列数据记录;

(3)任务实例(Task_instance),能够对数据执行如过滤、合并、统计、解析和查询数据库等操作,是实时流计算资源分配和执行任务的最小单位;

(4)计算节点(Bolt),一个计算节点可以包含多个任务实例,并且每个计算节点在系统中有一个唯一的地址,一个计算节点由多台物理主机构成,大多数实时流计算系统以一台主机作为控制,任务实例则分散在不同的物理主机上;

(5)监督模块(Watcher),监督模块用于监视系统中各模块工作情况,目前大部分实时流计算系统采用 ZooKeeper 开源系统^[2];

2.2 实时流计算编程模型

Storm 集群采用拓扑(Topology)编程模型(见图 2),拓扑编程模型提供统一的可复用的接口,支持多语言和分布式并行计算,提供可靠的数据处理,用户提交程序后系统会自动分配资源进行并行处理同时监控运行情况. 拓扑编程模型(TopologyBuilder)由两部分组成:数据源节点(Spout)和计算节点(Bolt). 数据源节点(Spout)是 Storm 系统中的消息生产者,通常情况下 Spout 会从外部读取数据并向内部发送消息,Spout 可以提供可靠和不可靠的消

息. Spout 节点存在的意义最主要意义是对外封装对内提供一个可以统一控制的数据来源. 计算节点(Bolt)可以初始化多个任务实例,用来执行对数据流的操作,系统中可以存在多个 Bolt,每个 Bolt 执行不同的操作,如过滤、聚合、查询数据库等等.

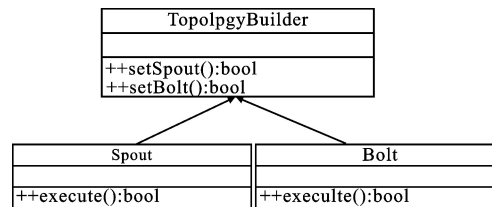


图 2 Topology 编程模型接口

3 基于 Storm 系统的 AOS 协议数据处理

国际上通常采用高级在轨协议(AOS)^[4]作为空间科学实验的下行数据协议,AOS 协议具有嵌套层次多、解析复杂、数据帧之间具有强关联性等特点,是航天系统中处理最复杂最具有代表性的一类数据,提高 AOS 在轨协议的解析速率具有非常重要的意义. 如图 3 所示为 AOS 协议的一个简单示例.

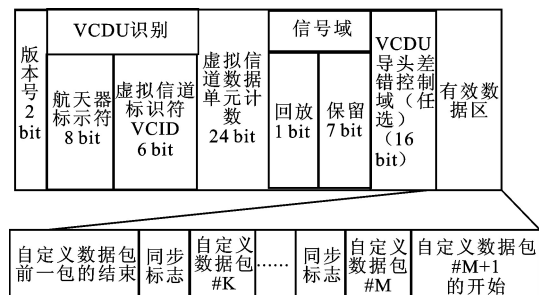


图 3 AOS 有效载荷下行数据参考格式

采用 Storm 系统处理 AOS 协议数据,能够非常方便地在虚拟信道层进行并行处理,使用者只需要实现 Spout 和 Bolt 中执行部分,选择数据发送方式为可靠,系统会自动进行并行部署,其中并行部分体现在每个 Bolt 中初始化的任务实例个数,这种并行是多物理主机多进程级别的,如图 4 所示为 AOS 编程的物理实现. Storm 系统中 Spout 为统一的数据源,即可以处理实时数据也可以处理文件、数据库等,对文件处理时是 Hadoop 集群的一种延续,可以复用现有地面系统中 Hadoop 大数据处理里的处理逻辑和部分代码.

此外,Storm 系统具有非常高的吞吐能力和容错性能^[5],在 CPU 利用率和消息传递方面,均具有非常优秀的表现,非常适合航天地面系统中用户服务、订单处理、消息查询等任务的处理^[6].

AOS 数据处理的 Storm 拓扑编程示例代码如下:

```
TopologyBuilder builder=new TopologyBuilder();
//新建一个拓扑程序结构
builder.setSPout(1,new AOSSpout("225.9.9.9",22133,"
data_stream",new StringScheme()));
//添加数据源节点,从 IP225.9.9.9 端口 22133 获取原始数据
builder.setBolt(2,new VCIDBolt(),1).shuffleGrouping(1);
//添加虚拟信道划分计算节点,只初始化一个任务实例
for each(string xmlPath in FilePath){
//根据不同的配置文件,初始化多个虚拟信道处理节点,每个节点只初始化一个任务实例
    builder.setBolt(3,new PayLoadBolt(xmlPath),1),fields-
    Grouping(2,new Fields("frame"));}
public class VCIDBolt implements IBasicBolt{
//VCIDBolt 继承自 IBasicBolt 用从原始信道中划分不同虚拟信道
    public void execute(Tuple trple, BasicOutputCollector collector){
//根据不同虚拟信道发送给不同有 PayloadBolt 进行处理
        swich(VCID){...case Oxee; collector.emit(new
        Values(frame, subString(8,500));}
    }
}
public class PayLoadBolt implements IBasicBolt{
//PayLoadBolt 载荷处理节点缓存达到数据帧逐个解析出
    自定义数据
    public PayLoad(string xmlPath){...} //初始化解析配置
    文件
    public void execute(Tuple trple, BasicOutput Collect or
    collector){
        ...//解析数据帧}}
}
```

4 拓扑模型的改进

以 Storm 系统为例,实时流计算系统在处理带有关联性数据流时存在瓶颈,本节将会论述通过并行的方式提高处理速度,并对处理效率进行分析。

4.1 时序数据帧并行处理

实时流计算最初的设计目标是用来处理网络请求、查询等独立的事务集^[3],并不适合于处理关联性数据流。以 Storm 系统为例,实时流计算是通过多个任务实例并行执行以减少处理时延,通常情况下要求数据流中的每个数据单元都能被单独处理,如果前后两个数据单元具有关联性,而这两个单元又被分在不同的任务实例中,那么这两个数据单元均无法正常解析。

如图 4 中 AOS 拓扑编程模型,每个计算单元中

就只初始化了一个任务实例。这样做虽然从虚拟信道的角度上实现了数据处理的并行,但是无法继续做更加细致的并行,无疑降低了计算性能^[7]。

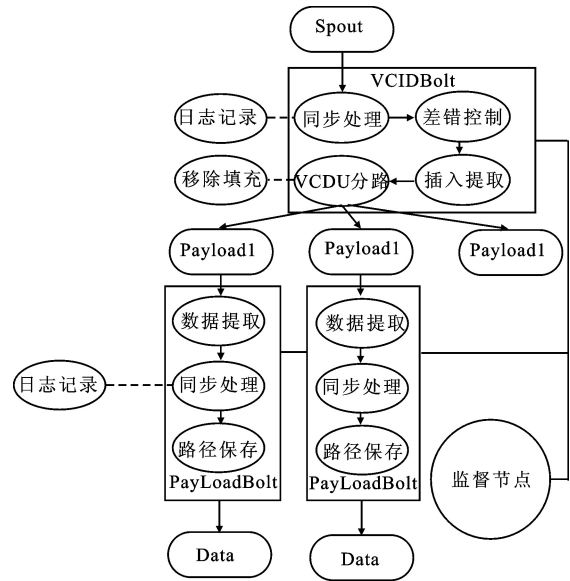


图 4 AOS 拓扑编程结构

本文通过构造多级并行计算网络来处理具有关联关系的数据流。处理流程图如图 5 所示。其中第一级计算节点 1 中有 5 个任务实例用于并行计算,如果能够独立解析,则直接输出,否则将当前数据帧按一定的映射关系向后发送,第二级计算节点 2 有两个任务实例,每个任务实例对达到数据帧排序后解析,如果能够独立解析则直接输出,否则继续传递给后续计算节点,第三级计算节点 3 中只有一个任务实例。最坏情况下如果所有数据帧均相关,所有数据帧都会达到计算节点 3,网络退化为单机模型,这样可以保证所有数据均能够被正确解析。在数据帧向后传递时可以采用一致性 Hash^[8]算法作为网络映射。

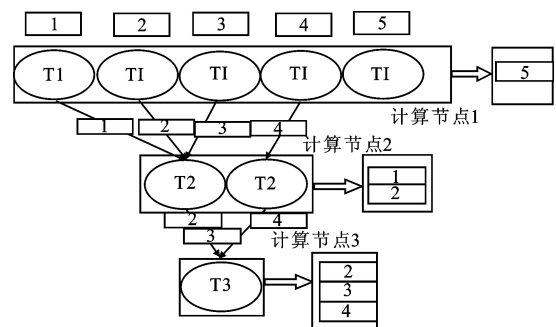


图 5 时序关联数据流内部并行方式

4.2 并行计算性能分析

根据前文分析可以知道,在解析 AOS 协议数据帧时影响并行计算的最大的因素,就是数据流中必须串行处理的数据帧所占的比重^[9]。首先假设所

有的最外层数据帧长度均相等,程序检测数据帧是否能够被完整解析,自定义数据帧解析时间 T 是最外层数据帧个数 n 的线性函数.

单机响应时间为

$$T(i) = P + \sum_{i=1}^m S_i = \text{Sum}, i = 1 \cdots m \quad (1)$$

式中, Sum 表示待处理的数据流中所有数据帧的个数, P 表示能够被单独解析的数据帧个数, S 表示必须串行解析的数据帧, S_i 表示第 i ($i = 1 \cdots m, m \in \mathbb{N}$) 个必须被串行解析的数据帧,且 $P + \sum_{i=1}^m S_i = \text{Sum}$

并行计算网络多机处理响应时间很难衡量,因为同时受到并行计算网络部署和数据流分布的影响. 首先分析只有两级的并行计算网络响应时间,其中第一级计算节点中有 N 个任务实例,第二级计算节点中只有一个任务实例, Q 为网络传输时延, $N-1$ 多机响应时间为

$$T(N-1) = \max\left\{\frac{P}{N}, \sum_{i=1}^m S_i\right\} + m * Q, i = 1 \cdots m \quad (2)$$

式中, $\frac{P}{N}$ 表示第一级任务实例处理所花费的时间, $\sum_{i=1}^m S_i$ 表示第二级任务实例的处理时间,由于两部分操作是并行的,所以取最大处理时间.

当后续处理节点存在多级多个任务实例时,串行数据帧也能够被并行处理,如果网络映射固定,并且数据流分布均匀,可以采用递归的思想来衡量网络的处理速度. N_i 表示第 i 级并行网络有 N_i 个任务实例, $T(N_i + 1)$ 表示 N_i 之后的并行处理网络响应时间,对于多级并行计算网络:

$$T(N_i) = \max\left\{\frac{P_i}{N_i}, T(N_i + 1)\right\} + m * Q, i = 1 \cdots m \quad (3)$$

加速比:

$$S_n = \frac{T(1)}{T(N)} = \frac{P + \sum_{i=1}^m S_i}{\max\left\{\frac{P_i}{N_i}, T(N_i + 1)\right\} + m * Q} \quad (4)$$

从加速比中可知,极限情况下当所有数据帧均可被独立解析式时,加速比最大,当所有数据帧相关时加速比最小,退化到单机处理速度.

5 仿真算例

数据帧并行处理原型仿真实验硬件环境为: Intel i5-2400 CPU 主频 3.10 GHz, 4 GB 内存, 局域网带宽为 1 Gb/s, 软件环境为: 操作系统 win7, 开发环

境为 Visual studio 2010. 并行数据处理网络为 C# 开发的模拟原型, 网络协议采用 UDP, 局域网内部有 3 台主机, 其中 2 台主机为前端处理主机, 1 台主机为后续处理主机, 每台主机运行 4 个任务实例. 数据来源为图 3 所示的 AOS 格式协议模拟生成数据, 数据帧格式严格按照图示格式, 其中每个自定义数据包长度是随机产生的, 但不会超过 8096 字节, 每个 UDP 数据包长度为 1024 字节. 接收端处理方式: 先缓存到达数据帧, 然对到达数据帧按帧序号排序. 并行处理时, 并不改变单机处理代码, 只是当处理端接收到数据帧后, 首先判断当前数据帧是否能够被处理, 如果无法解析则发送给后续处理节点.

根据式(3)可以估算出并行计算网络的响应时间和处理能力, 在增加一定的网络传输延迟后, 处理速度曲线如图 6 所示, 约定(4-3-1)表示: 第一级计算节点中有 4 个任务实例, 第二级计算节点中有 3 个任务实例, 第三级计算节点中只有一个任务实例, 局域网中有 2 台主机, 8 个处理进程. 图中点表示为实际测量处理速率, 误差在 5% 以内.

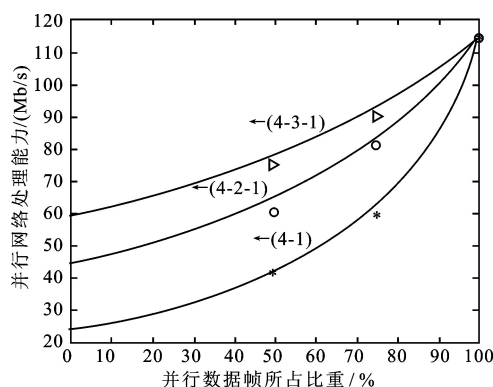


图 6 并行理论处理效率曲线

从实验结果(表 1)可以看出, 采用 4 个前端任务实例和 1 个后续任务实例时, 当可以独立解析的数据帧所占比重较大时, 处理速率较高, 当关联性数据帧所占比重较高时, 由于大量的处理由第二级 1 个进程承担, 处理效率降低, 而采用 4-3-1 处理网络时, 第二级有 3 个进程并行处理, 极大提高了并行处理效率. 而通常空间科学数据会有一定的规律性, 如大约连续 K 个数据帧能够解析出一幅完整图像等, 所以可以通过增加第一层每个任务实例处理的数据帧个数, 使得每个任务实例能够解析出一幅完整数据图像, 在日常处理中这种方式能够成倍的提高处理速率.

6 结束语

实时流计算系统能够有效地处理目前航天地面

系统中大部分数据类型,具有非常强的可扩展性和应用性,能够为用户服务、订单处理等实时任务提供高速的响应,并且支持大数据处理,能够在后续工作中为数据挖掘等科学研究提供有效的数据处理方式。虽然航天数据自身结构非常复杂而且实时性要求较高,目前在高速处理方面还有所欠缺,但是在将来,实时流计算概念一定会逐步成为航天地面系统数据处理的主要方式。

表1 并行计算处理结果 Mb/s

	单机处 理速率	并行处理 速率(4- 1)	并行处理 速率(4-2- 1)	并行处理 速率(4-3- 1)
单帧事务数据流	30	120	120	120
单帧事务集比重 75%	30	60	82	91
单帧事务集比重 50%	30	42	61	76
随机长度数据帧	30	44	62	80

参考文献:

- [1] 李博权. 空间多载荷协同工作信息可视化研究[J]. 微电子学与计算机, 2012, 29(8): 56-58.
- [2] Tom White. Hadoop 权威指南[M]. 2版. 北京:清华大学出版社, 2011.
- [3] Apache Incubator. Storm-wiki [EB/OL]. (2013-12-01). [2014-06-07]. <http://storm-project.net>.

(上接第14页)

参考文献:

- [1] Liedtke J. Improving IPC by kernel design[C] // ACM SIGOPS Operating Systems Review. Germany: Dags-tuhl Csstle, ACM, 1994; 27(5): 175-188.
- [2] Blackham B, Shi Y, Heiser G. Improving interrupt response time in a verifiable protected microkernel[C] // Proceedings of the 7th ACM european conference on Computer Systems. Swizerland, Berne, ACM, 2012: 323-336.
- [3] Dannowski U, LeVasseur J, Skoglund E, et al. I4 experimental kernel reference manual [R]. Germany: University of Karlsruhe, Version X, 2011.
- [4] Blackham B, Heiser G. Correct, fast, maintainable; choose any three! [C] // Proceedings of the Asia-Pacific Workshop on Systems. China: Beijing, ACM, 2012: 13.
- [5] Marzi H, Yang Y. Microkernel based real-time embedded operating systems efficiency improvement [C] // Electrical & Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on. Montreal, IEEE, 2012: 1-4.

- [4] 左莉华. 高级在轨系统(AOS)数据传送与分路简介[J]. 遥测遥控, 1993(3): 47-49.
- [5] 量子恒道. Storm 系统框架研究[EB/OL]. (2013-12-10). [2014-06-07]. <http://blog.linezing.com>.
- [6] Ghemawat S, Gobioff H. The google file system[C] // Proceeding of the 19th ACM Sympoium on Operating Systems Principles. New York: ACM Press, 2003.
- [7] 刘伟峰, 王智广. 细粒度并行计算编程模型[J]. 微电子学与计算机, 2008, 25(10): 72-74.
- [8] David karger, Eric lehman. Consistent hashing and random trees[C] // STOC'97 Proceedings of the twenty-ninth annual ACM symposium on Theory of Computing. New York, 1997.
- [9] Frank Dehne. Scalable parallel computing geometry [C] // Michel Cosnard, Afonso Ferreira, Parallel and Distributed Computing. USA: Afonso Ferreira, 1994.

作者简介:

李一辰 男, (1987-), 硕士研究生. 研究方向为实时流计算.
李绪志 男, (1968-), 研究员, 博士生导师. 研究方向为地面应用系统总体设计、综合信息系统、智能信息处理.
阎镇 男, (1974-), 硕士, 高级工程师. 研究方向为空间科学与应用科学数据可视化.

作者简介:



张凡 男, (1979-), 博士, 副教授. 研究方向为嵌入式计算技术、形式化方法.



王晓鹏 男, (1988-), 硕士研究生. 研究方向为嵌入式计算技术、形式化方法.



苏卫宁 男, (1990-), 硕士研究生. 研究方向为嵌入式计算技术、形式化方法.