arXiv:2301.03734v1 [cs.DC] 10 Jan 2023

Exoshuffle-CloudSort

FRANK SIFEI LUAN*，加州大学伯克利分校STEPHANIE WANG，加州大学伯克利分校和 AnyscaleSAMYUKTA YAGATI，加州大学伯克利分校SEAN KIM，加州大学伯克利分校KENNETH LIEN，加州大学伯克利分校ISAAC ONG，加州大学伯克利分校TONY HONG，加州大学伯克利分校SANGBIN CHO，AnyscaleERIC LIANG，AnyscaleION STOICA，加州大学伯克利分校和 Anyscale

1 引言

我们介绍 Exoshuffle-CloudSort，这是一款基于 Ray 并采用 Exoshuffle 架构运行的排序应用程序 [4]。Exoshuffle-CloudSort 运行于 Amazon EC2，输入和输出数据存储在 Amazon S3 上。使用 40× i4i.4xlarge 个工作节点，Exoshuffle-CloudSort 在 5378 秒内完成了 100TB CloudSort 基准测试（Indy 类别 [6]），平均总成本为 97 美元。

2 实现

2.1 概述

Exoshuffle-CloudSort 是一个运行在 Ray 之上的分布式 futures 程序。Ray 是一个基于任务的分布式执行系统。该程序充当控制平面，用于协调 Map 和 Reduce 任务；Ray 系统则充当数据平面，负责执行任务、传输数据块以及从故障中恢复。

Exoshuffle-CloudSort 实现了一种两阶段外部排序算法。第一阶段是 Map 和 Shuffle。每个 Map 任务读取一个输入分区，对其进行排序，并将结果划分为W个输出分区，每个分区都会发送到工作节点上的一个合并器。合并器接收W个 Map 输出分区，将它们合并并排序，然后进一步将结果划分为 R/W 个输出分区，所有分区都会溢写到本地磁盘。

第二阶段是 Reduce。Map 和 Shuffle 阶段结束后，每个 Reduce 任务读取W个 Shuffle 分区，将它们合并并排序，然后写入最终输出分区。

对于 100 TB CloudSort 基准测试，我们设置了以下参数：

- 数据总计大小为 100 TB。
- 输入分区数量为 $M = 50\,000$。每个输入分区为 2 GB。

*作者地址：lsf@berkeley.edu，美国加利福尼亚州伯克利市 Soda Hall 465 室。

- 工作节点数量 $W = 40$。
- 输出分区数 $R = 25\,000$。

2.2 准备

Exoshuffle-CloudSort 的第一步是计算分区边界值。对于键长为 10 字节的排序记录，我们将前 8 个字节视为一个 64 位无符号整数分区键。我们将键空间$[0, 2^{64} - 1)$划分为 $R = 25\,000$ 个相等范围，使每个键范围内的所有记录都发送到一个归约器。

每 $R_1 = R/W = 625$ 个归约器范围合并为一个工作节点范围，每个工作节点范围内的记录都将发送到一个工作节点。这会产生 $W = 40$ 个均等划分的工作节点范围。

2.3 映射和洗牌阶段

在映射与混洗阶段，Exoshuffle-CloudSort 会将 $M = 50\,000$ Map 任务调度到所有工作节点上。在实验中，我们将映射并行度（即单个工作节点上运行的 Map 任务数）设为虚拟 CPU 核心总数的 3/4。额外的任务会排入驱动节点上的队列。每当某个工作节点完成一个 Map 任务，驱动节点就会从队列中分配一个新任务给该节点。

在 Map 任务中，我们首先从 S3 下载输入分区。然后在内存中对输入数据进行排序，再将其划分为 $W = 40$ 个切片。每个切片都会立即发送到各工作节点上的合并控制器。所有切片发送完毕后，Map 任务即返回。

在接收端，合并控制器会将映射块累积在内存中，直到达到阈值。我们将阈值设为 40 个数据块，即约 2GB 数据。达到阈值后，控制器会启动一个合并任务，将已排序的映射块合并，并进一步将其划分为 $R_1 = 625$ 个合并块，每个合并块对应此节点上的一个归约任务。这些数据块会溢写到本地固态硬盘，供归约器使用。

合并并行度设置为与映射并行度相同。当合并任务数达到最大并行度，且合并控制器的内存缓冲区已满时，它会暂缓确认收到映射数据块，直到某个合并任务完成并可以启动新的合并任务。这实际上会对映射任务调度器施加反压，以确保映射、混洗和合并进度保持同步。

在我们的实验中，映射任务的平均耗时为 24 秒，其中 15 秒用于下载输入数据。平均混洗时间（即发送和接收数据块的时间）为 7 秒。合并任务的平均耗时为 17 秒。

2.4 Reduce 阶段

所有 Map 和合并任务完成后，Exoshuffle-CloudSort 进入归约阶段。每个归约任务从本地固态硬盘加载$R_1 = 625$，将其合并，并把排序后的输出分区上传到 S3。在我们的实验中，每个归约任务平均耗时 22 秒。

2.5 执行系统

Exoshuffle 架构的一大亮点是，应用程序只需实现控制平面逻辑，而执行工作则由分布式期货系统 Ray 负责。这一点在 Exoshuffle-CloudSort 中得到了体现。以下是不完全列举的 Ray 所提供、我们可以“免费”使用的功能：

- 任务调度：程序指定任务的调度时间和位置；系统负责处理远程过程调用、序列化及其他簿记工作。
- 网络传输：程序通过将分布式 Future 作为任务参数来指示数据传输；系统负责实现高性能网络传输。
- 内存管理和磁盘溢写：程序在虚拟的无限地址空间中操作数据引用；系统通过引用计数管理分布式内存，在内存不足时将对象溢写到本地磁盘，并在需要时从本地磁盘恢复对象。
- 网络和磁盘 I/O 的流水线处理：对象的网络传输、溢写和恢复对应用程序透明，并以异步方式执行。例如，系统会在其他 Map 和合并任务运行时对 Map 输出块进行 shuffle；在其他合并任务执行时将合并任务的输出溢写到磁盘，并在 Reduce 任务执行时恢复合并块。
- 容错：这对应用程序是透明的；系统遇到网络故障或工作进程故障时，会自动重试该操作。

更多详情，请参阅 Ray 架构白皮书 [7],中关于分布式 Future 系统的所有权设计的介绍 [8], 以及 Exoshuffle 论文 [4]。

2.6 源代码

Exoshuffle-CloudSort 使用约 1000 行 Python 代码和约 300 行 C 代码实现++。C++ 组件实现了两项功能：排序和记录分区，以及合并已排序记录数组。Exoshuffle-CloudSort 运行于 Ray 之上，而 Ray 使用 Python 和 C++实现。Exoshuffle-CloudSort 的所有源代码均可在以下网址获取：https://github.com/exoshuffle/cloudsort。

3 评估

3.1 环境设置

我们在 AWS 上的计算集群中运行 Exoshuffle-CloudSort，集群配置如下：

- 1× r6i.2xlarge 主节点。该节点配备 8 核、主频为 2.9GHz 的英特尔至强 8375C 处理器，以及 64 GiB 内存。
- 40× i4i.4xlarge 工作节点。每个节点配备 16 核、主频为 2.9 GHz 的英特尔至强 8375C 处理器，以及 128 GiB 内存。每个节点还配备一块直接连接的 3.75TB AWS Nitro NVMe 固态硬盘。
- 每个节点还连接了一个 40 GiB 的 Amazon EBS 通用型固态硬盘 (gp3) 卷。

软件栈配置如下：

- Ubuntu 22.04.1 LTS，Linux 内核版本为 5.15.0-1022-aws。
- XFS 5.13.0 文件系统。
- Intel oneAPI DPC++/C++ 编译器 2022年2.0.20220730。
- Python 3.9.13。
- Ray 2.1.0。

我们使用标准基准测试工具测量工作节点上的原始系统 I/O 性能：

- 网络带宽：节点间为 25 Gbps，使用 iperf 进行基准测试。
- 固态硬盘：读取速度为 2.9 GB/s，写入速度为 2.2 GB/s，使用 fio 进行基准测试。

对于存储，我们在 Amazon S3 上使用 40 个存储桶，并将输入分区和输出分区随机分配到这些存储桶中。

3.2 基准测试设置

生成输入。我们使用 `gensort` Sort Benchmark 委员会提供的 1.5 版本 [5]。我们运行命令 `gensort -c -b{offset} {size} {path}` 来生成每个分区。`{size}` 固定为$P = 20\,000\,000$，使每个分区恰好为 2 GB。`{offset}` 取值为 $\{i \cdot P\colon 0 \le i < M\}$，其中输入分区的数量为 $M = 50\,000$。`{path}` 是 tmpfs 中的唯一路径。`-c` 用于生成数据校验和以进行验证。生成输入文件后，我们随机选择一个存储桶，并将该分区上传到 S3。我们使用 Ray 将 50 000 个输入生成任务调度到全部 40 个工作节点。生成的结果汇总为输入清单文件，并保存起来，供 Exoshuffle-CloudSort 用于定位排序输入。

验证输出。 Exoshuffle-CloudSort 会生成一个输出清单文件，其中包含 S3 上每个输出分区的存储桶和键。在每个验证任务中，我们先将输出分区下载到 tmpfs，然后运行命令 `valsort -o {sumpath} {path}` 来验证各分区中的记录顺序。我们使用 Ray 将 25 000 个输出验证任务调度到全部 40 个工作节点。我们将各验证任务生成的汇总文件内容拼接起来，然后运行 `valsort -s` 以验证总体排序，并生成总输出校验和。最后，我们将输出校验和与输入校验和进行比较，以验证数据完整性。

3.3 实验结果

3.3.1 作业完成时间。 2022年11月10日，我们在 AWS 美国西部（俄勒冈州，us-west-2）区域，按照上述配置运行了 100TB CloudSort 基准测试。我们首先在 Amazon S3 上生成输入数据，然后运行 Exoshuffle-CloudSort 3 次，每次运行后都执行验证步骤。3 次运行均成功，且输出校验和与输入校验和一致，表明排序过程中所有字节均得到保留。表 1 报告了每次运行的作业完成时间。平均作业完成时间为 5378 秒，即 1.4939 小时。

图 1 展示了在 100TB CloudSort 基准测试的第 #1 次运行期间，计算集群中所有工作节点的系统利用率。

Run	映射与洗牌时间	归约时间	作业总完成时间
#1	3509 s	1852 s	5361 秒
#2	3496 秒	1852 秒	5348 秒
#3	3520 秒	1906 秒	5426 秒
平均值	3508 秒	1870 秒	5378 秒

表 1. Exoshuffle-CloudSort 在 100TB CloudSort 基准测试中的作业完成时间。



图 1. 100TB CloudSort 基准测试第 1 次运行期间的集群利用率。每条粗线表示所有工作节点的系统利用率中位数；最高线和最低线分别表示所有工作节点中的最大利用率和最小利用率。

3.3.2 总拥有成本。 作业总成本由两部分构成：计算成本（Amazon EC2）和存储成本（Amazon S3）。存储成本又进一步分为数据存储成本和数据访问成本。

计算成本。 计算成本按计算集群的每小时成本乘以作业完成时间计算。每小时计算总成本的计算方式如下：

$$\begin{aligned}\text{每小时计算总成本} &= \text{主节点每小时成本} + \text{工作节点每小时成本} \times \text{工作节点数量} \\ &+ \text{EBS 卷每小时成本} \times (\text{工作节点数量} + 1)\end{aligned} \tag{1}$$

我们从亚马逊 EC2 按需定价信息中获取计算实例每小时成本 [2]。对于 EBS，我们使用 Amazon EBS 月度价格 [1]除以一个月的平均小时数（$\frac{365\times24}{12} = 730$），作为每小时价格。一个 40 ~~GiB gp~~3 卷的每小时成本为 $\$0.08/730 \times 40 = \$0.0044$。现在，我们将成本变量代入公式（1）：

- 主节点（r6i.2xlarge）的每小时成本为 $0.504。
- 工作节点（i4i.4xlarge）的每小时成本为 $1.373。
- 工作节点数量为 40。
- EBS 卷的每小时成本为 $0.0044。

因此，每小时计算总成本为 $55.6044。我们将此每小时成本乘以 1.4939 小时的作业完成时间，得出计算总成本为 $83.0674。

数据存储成本。 存储成本包括数据存储成本和数据访问成本。我们首先考虑数据存储成本。Amazon S3 采用按需付费定价模式，即用户无需提前预留存储容量，只需根据对象的大小和存储时长支付存储费用。Amazon S3 对前 50 TB 按每 GB-月 $0.023 收费，接下来的 450 TB 则按每 GB-月 $0.022 收费 [3]。由于数据总量为 100 TB，我们采用前两个档位的平均价格，即每 GB-月 $0.0225，或每 100 TB 每小时 $3.0822。

- 输入：100 TB 输入数据的存储成本，就是在排序期间存储 100 TB 数据的成本：$3.0822 × 1.4939 = $4.6045。
- 输出：100 TB 输出数据会在排序的归约阶段上传并存储到 Amazon S3。我们将归约阶段的持续时间作为 100 TB 输出数据的存储时间。这是一个偏高的估算，因为输出分区会随着归约阶段的推进而逐步上传，因此大部分 100 TB 数据在 S3 上的存储时间都短于整个归约阶段的持续时间。表 1 显示，归约阶段的平均耗时为 1870 秒，即 0.5194 小时。因此，输出数据的存储成本为：$3.0822 × 0.5194 = $1。6009。

将输入和输出数据的存储成本相加，可得数据存储总成本：$6.205 4.

数据访问成本。我们考虑对 Amazon S3 的 GET 和 PUT 请求。Exoshuffle-CloudSort 在 50 000 个 Map 任务中下载 100 TB 输入数据。每个 Map 任务以 16 MiB 为单位下载一个 2 GB 输入分区，因此每个任务产生 120 个 GET 请求，总计 6000 000 个 GET 请求。Amazon S3 对每 1000 个 GET 请求收费 $0.0004 [3]。因此，GET 请求的总成本为 $2.4000。

Exoshuffle-CloudSort 在 25 000 个 Reduce 任务中上传输出数据。每个 Reduce 任务以 100 MB 为单位上传约 4GB 数据，因此产生 40 个 PUT 请求，总计 1 000 000 个 PUT 请求。Amazon S3 对每 1000 个 PUT 请求收费 $0.005 [3]。因此，PUT 请求的总成本为 $5.0000。

由于请求失败和重试，实际请求数量可能略高，但增加的数量可以忽略不计。因此，数据访问成本总计为 $7.4000。

总拥有成本。将计算成本和存储成本相加，我们得出 100TB CloudSort 基准测试的总拥有成本：$96.6728。表格2 汇总了成本分析结果。

服务	单价	数量	总价
计算虚拟机集群	$55.6044 / 小时	1.4939 小时	$83.0674
数据存储（输入）	$3.0822 / 小时	1.4939 小时	$4.6045
数据存储（输出）	$3.0822 / 小时	0.5194 小时	$1.6009
数据访问（输入）	$0.0004 / 1000 次请求	6 000 000 次请求	$2.4000
数据访问（输出）	$0.005 / 1000 次请求	1 000 000 次请求	$5.0000
总计	–	–	$96.6728

表 2. Exoshuffle-CloudSort 在 100TB CloudSort 基准测试中的成本明细。

致谢

这项工作在加州大学伯克利分校的 Sky Computing Lab 完成，由 Astronomer、Google、IBM、Intel、Lacework、Nexla、Samsung SDS 和 VMware 赞助。这项工作与 Anyscale 合作完成。

参考文献

[1] 亚马逊。2022年。Amazon EBS 高性能块存储定价。亚马逊云科技。https://aws.amazon.com/ebs/pricing/[2] 亚马逊。2022年。Amazon EC2 按需实例定价。亚马逊云科技。https://aws.amazon.com/ec2/pricing/on-demand/[3] 亚马逊。2022年。Amazon S3 简单存储服务定价。亚马逊云科技。https://aws.amazon.com/s3/pricing/[4] FRANK SIFEI L UAN、STEPHANIE WANG、SAMYUKTA YAGATI、SEAN KIM、KENNETH LIEN、ISAAC ONG、SANGBIN CHO、ERIC LIANG 和 ION STOICA。2022年。Exoshuffle：应用层的大规模数据混洗。https://doi.org/10.48550/ARXIV.2203.05072[5] Chris Nyberg。2022年。排序基准测试数据生成器和输出验证器。Ordinal Technology Corp. http://www.ordinal.com/ gensort.html[6] Mehul A. Shah、Amiato 和 Chris Nyberg。2014年。CloudSort：总拥有成本排序基准测试。http://sortbenchmark.org/2014_06_CloudSort_v_0_4.pdf。（访问日期：2022年11月10日）。[7] Ray 团队。2022年。Ray v2 架构。Anyscale。https://docs.google.com/document/d/1tBw9A4j62ruI5omIJbMxly-la5w4q_TjyJgJL_jN2fI/preview[8] Stephanie Wang、Eric Liang、Edward Oakes、Ben Hindman、Frank Sifei Luan、Audrey Cheng 和 Ion Stoica。2021年。Ownership：面向细粒度任务的分布式 Future 系统。载于第18届 USENIX 网络系统设计与实现研讨会（NSDI 21）。USENIX 协会，线上，671–686。https://www.usenix.org/conference/nsdi21/presentation/cheng