快速确定大数据 Spark 应用配置参数值域的方法研究

李瑞[1,2]，李乐乐[2]，喻之斌[2,3]

[1]（南方科技大学 深圳 518055）

[2]（中国科学院深圳先进技术研究院 深圳 518055）

[3]（中国科学院大学 北京 100049）

摘要：当前，互联网数据处理规模增长迅猛，用户的处理需求也变化多样。为适应不同数据处理场景的需求，如不同的集群资源和数据集，流行的大数据处理系统为用户提供了越来越多的可配置参数。譬如，最受欢迎的大数据处理系统 Spark 提供了超过 200 个以上的参数。它们控制了应用程序的并行度、I/O 行为、内存使用和数据压缩等。但是，这些配置参数设置不当往往会导致程序性能严重下降，甚至导致大数据系统的运行崩溃，给客户造成不可估量的损失。

为了合理设置参数值域，保证大数据应用的安全运行和高性能，本文提出了一种名为复合搜索的方法，确定了与大 Spark 应用程序性能密切相关的 38 个配置参数的值域。首先，我们以 Spark 配置参数默认值为起点，分别采用初始区间探测和无目标值搜索，确定其值域的初始范围，然后进一步对配置参数间的依赖关系进行分析，进一步缩小参数的有效值域范围，最终显著地放缩了参数空间。

我们选择一个包含 4 个 x86 节点的集群，并使用全部 103 个 TPC-DS SparkSQL queries 来评估复合搜索的效果。实验结果表明，复合搜索相比传统方法分别在程序和参数维度上值域搜索的加速比为平均 5.5 倍和 4.9 倍，并且，复合搜索找到的参数值域使得程序运行的平均成功率由原来的 46.5%提升到 81.7%，在现有的实验驱动调优和机器学习调优方法下应用复合搜索，能够平均减少 30%的时间开销。

关键词：Spark；配置参数；值域范围；复合搜索；依赖关系

Research on the Accurate and Fast Determination Method of Spark Configuration Parameter Range

Rui Li[1,2],Lele Li[2], Zhibin Yu[2,3]

[1](Southern University of Science and Technology, Shenzhen 518055, China)
[2](Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China)
[3](University of Chinese Academy of Sciences, Beijing 100049, China)

Corresponding Author：Zhibin Yu E-mail：zb.yu@siat.ac.cn

Abstract: Currently, with the exponential growth of data on the internet, the complexity of big data processing systems has also increased dramatically. To adapt to changes in factors such as cluster resources, datasets, and applications, big data processing systems provide adjustable configuration parameters tailored to different application scenarios. Among these systems, Spark is one of the most popular and contains over 200 configuration parameters for controlling parallelism, I/O behavior, memory settings, and compression. Incorrect configuration of these parameters often leads to severe performance degradation and stability issues. However,

---

both ordinary users and expert administrators face significant challenges in understanding and tuning these settings for optimal performance, resulting in substantial human and time costs. In the tuning process, selecting unreasonable parameter ranges can increase time costs by fivefold, or even worse, cause operational failures in the cluster and terminate system operation—an incalculable loss for large-scale clusters serving customers.

To address this issue, this paper proposes a compound search method, named CS, to determine the value ranges of 38 configuration parameters that are closely related to the performance of SparkSQL applications. CS first categorizes the configuration parameters into nine groups based on their functional attributes, then models dependencies among related parameters within each group. Starting from the default values of Spark's configuration parameters, CS probes and defines the value ranges of 38 key parameters, significantly compressing the parameter space.

To be more general, we chose the SparkSQL application to evaluate CS using all 103 queries in TPC-DS in a cluster of 4 x86 nodes. Experimental results show that compared with the traditional methods, the performance of composite search is improved by an average of 5.5 times and 4.9 times in the program dimension and parameter dimension, respectively, and the success rate of Spark SQL program running in the value range of composite search is increased from an average of 46.5% to an average of 81.7%, which is of great significance for the application and optimization of Spark SQL program.Applying composite search under existing experiment-driven tuning and machine learning tuning methods can reduce time overhead by an average of 30%.

Key words: Spark;Configuration Parameters;Value Range;Compound search

1 引 言

当前，万维网、电子商务、物联网（IoT）和其他应用程序的持续增长每天都会产生大量不断增加的原始数据。例如，在视频软件 Youtube 上，每月有超过 60 亿小时的视频播放量，其数据大小达到了 263PB[1,2,3,4]。每天有超过 100TB 的数据在 Facebook 完成上传，Facebook 的总数据大小约为 252PB。为有效地收集、处理和分析大量数据，许多大数据分析平台被开发和广泛使用，包括批处理数据处理系统（如 Hadoop，MapReduce，Spark）和流数据处理系统（如 Apache Storm，Heron，Flink，Samza）。在如此大的规模上实现良好而稳健的系统性能是成功执行及时且经济高效的分析的基础[5,6,7]。然而，系统性能与大量配置参数直接相关，这些参数控制着系统执行的各个方面，从低级内存设置和线程计数到高级决策，如资源管理和负载平衡。配置参数设置不当会对整体系统性能和稳定性产生不利影响[8,9]。

无论使用哪种平台，数据分析师和系统管理员经常难以为其应用程序找到适当的参数设置，因此他们通常依靠直觉、经验、数据领域知识以及其他专家的最佳实践建议或调优指南来调优其应用程序。目前面临的主要挑战是：（1）巨大而复杂的参数空间：Spark 有超过 200 个可配置参数，并且某些参数可能以不同的方式影响不同作业的性能，同时某些参数组可能具有依赖效应，即一个参数的最优设置可能取决于另一个参数的设置[10]。（2）系统规模和复杂性: 随着数据分析平台在规模和复杂性上的增长，系统管理员可能需要配置和调整数百到数千个节点，其中一些节点配备了不同的 CPU、内存、存储介质和网络堆栈。此外，并行或串行执行具有迭代阶段和任务的 Spark 工作负载，使得观察和建模工作负载性能具有挑战性[10,11,12]。

参数调优问题定义如下，给定一个程序 p 在集群资源 r 上处理输入数据 d，找出使配

置空间 S 上 F 最大的最优配置参数设置 $c*$：

$$c* = \underset{c \in S}{\operatorname{argmax}} F(p,d,r,c) \quad (1)$$

实际情况中，性能函数 F 通常是未知的或部分已知的，而且相关的实验结果表明[13]，它是非凸的和多模态的。而且，在这种情况下寻找最优解是非常困难的。而我们的复合搜索就是在相同的 Spark SQL 程序 p，集群资源 r 和处理输入数据 d 上探测有效的配置空间 S 的。

该问题由 4 个部分组成，首先是应用程序 p，Spark SQL 应用程序类型种类多样，其中一个程序可能会包含一个或多个 query，这些 query 通常涉及对输入数据 d 的查找、聚合、排序等等 SQL 操作。而这些 query 所表现出的性能与 spark 配置参数密切相关。其次是集群资源 r，所指的是 Spark SQL 应用程序运行环境下可分配的集群资源，包括 CPU 核心数，内存大小，网络带宽等等[14]。然后，输入数据 d，通常由多个数据表组成，数据表主要分为两类，维度表和事实表，各个表的大小和结构分布往往不同。最后，配置空间 S，它表示在该应用场景下的配置参数的值域范围，具体指的是根据参数的功能属性，结合集群资源 r，输入数据 d 以及应用程序 p 为每个配置参数值的选取提供参考，从而压缩了参数空间，有效地防止程序运行中出现内存溢出，长时间挂住等严重系统故障。

我们可以把 Spark SQL 应用程序的配置空间探测抽象为一个数学问题。该问题的目标是在给定集群资源和输入数据的情况下，确定 Spark SQL 应用程序的配置参数的值域，期望达到的效果是能在尽量小的时间成本下得到比较精确的值域。概括来说，就是一个配置参数领域的时间与精度权衡下的搜索问题，由于 Spark 配置参数本身的定义和功能属性决定了绝大多数配置参数的值域为一个连续的且存在上下界的区间。并且 Spark 官方文档给出了每个配置参数的默认值。因此我们可以由默认值出发，探测 Spark SQL 配置参数的值域，这是显而易见的。

那么，我们如何探测每个配置参数的上下界呢？首先，确定边界的判定条件是非常有必要的。通常 Spark SQL 应用程序会因配置参数设置不合理而发生程序执行异常的状况，例如 Spark 执行器所分配的内存大小如果超出了集群资源可分配内存的大小，则会发生内存溢出的错误，而且当并行度设置不合理时，可能会出现严重的数据倾斜，从而导致 Spark SQL 作业执行时间过长，程序长时间挂住的状态。另外，如果网络带宽相关的配置参数设置不当，还会造成网络的拥塞，从而严重影响程序执行。因此可以把这些错误和异常情况归类为配置参数的边界判定条件，可用一个简单的约束函数表示。

得到了边界的约束条件，接下来需要确定搜索的起点，Spark SQL 官方文档给出了各个配置参数的默认值，这些取值绝大多数情况能够处于配置参数的值域范围内，因此我们以默认值为起点，将 Spark SQL 配置参数的值域搜索过程转化为一个约束条件下的搜索问题，即从固定点出发，在区间不确定以及目标值不确定的条件下，探测变量的上下界。这个问题的难点和关键就在搜索区间和目标值都不确定，一些常规的搜索方法如二分法均无法使用，并且搜索的过程不涵盖收敛的模块，这就导致了这个问题的本质变成了一个优化问题而不是表面上显而易见的搜索问题。

我们利用一种复合搜索的数学分析方法，结合分类法对 Spark SQL 应用程序的配置参数进行功能分区以及依赖关系的表征，对 SparkSQL 应用程序的配置参数的值域范围进行探测和确定，我们的主要贡献如下：

（1）我们实现了一种复合搜索（Compound Search，简称 CS）的数学分析方法，实现了给定起点，在非确定区间下进行区间的探测和边界的确定。

（2）采用了分类法对 Spark SQL 应用程序的配置参数进行了分类，根据其功能和属性，表征出了相关配置参数之间的依赖关系。

（3）结合分类法和复合搜索，我们可以求出应用场景下的 Spark 应用程序的配置参数值域范围，为配置参数的调优提供了参考值，以及收集训练样本各个配置参数合理的取值边界，从而减少错误样本，提高样本收集的质量，并且压缩了配置参数空间，能够增加机器学习调优的收敛过程，从而减少配置参数调优的时间成本。

我们在一个 4 节点的 ARM 集群和一个 8 节点的 x86 集群上对 TPC-DS 的查询进行了评估。实验结果显示，复合搜索相对传统方法在两种维度上都具有显著的性能提升，在配置参数维度上，值域搜索的速度提升平均 4.9 倍，在程序维度上提升平均 5.6 倍。另外，在相同的机器学习调优过程中，使用复合搜索的值域范围，能使训练样本收集阶段的程序运行成功率由平均 46.5%提升到平均 81.7%，节约了大量确定配置参数值域范围的时间成本，并且提高了程序运行的成功率。在现有的实验驱动调优与机器学习调优方法中应用复合搜索，能平均减少 30%的时间开销。

2 研究现状与动机

2.1 配置参数调优

目前许多研究通过部分或完全自动化在大数据处理系统中为执行任务寻找接近最优参数值的过程来解决性能优化问题[15,16]。现有的参数调优方法解决了高吞吐量和资源利用率、快速响应时间和成本效益方面的各种挑战。针对不同的挑战和场景，提出了不同的策略或方法。这些方法可分为以下六类:

（1）基于规则的方法帮助用户根据人类专家的经验、在线教程或调优说明来调优某些系统参数。它们通常不需要模型或日志信息，适合于快速引导系统[17,18,19]。

（2）成本建模方法通过基于对系统内部的深入理解而开发的分析(白盒)成本函数来构建高效的性能预测模型。建立模型通常需要一些实验日志和一些输入统计数据[20,21,22]。

（3）基于仿真的方法建立基于模块化或完整系统仿真的性能预测模型，使用户能够模拟不同参数设置或集群资源下的执行[23,24,25]。

（4）实验驱动方法在搜索算法和实际运行日志反馈的指导下，以不同的参数设置反复执行一个应用程序，即一个实验[26,27,28,29,30]。

（5）机器学习方法利用机器学习方法建立性能预测模型。这种方法通常将复杂系统作为一个整体来考虑，并且对系统内部几乎没有了解，也就是说将系统视为一个黑盒子[30,31,32,33,34,35]。

（6）自适应方法在应用程序运行时自适应地调整配置参数，即可以根据环境的变化根据各种方法调整参数设置。它们支持对特别的和长时间运行的应用程序进行调优[36,37,46,47]。

这些方法分为六大类：基于规则的、成本建模的、基于仿真的、实验驱动的、机器学习的和自适应调优的。这六种方法中的每一种都在一个或多个方面表现出色，有其独特的应用场景。在不了解参数调优的情况下，实验驱动的方法是最容易实现的方法。随着经验的积累，可以建立规则或成本函数来有效地提高系统性能。随着系统复杂性的增加，对小部件的仿真有助于增加对系统特性的理解。当尝试调整具有大参数空间的非常复杂的系统和应用程序时，具有适当训练数据的机器学习方法可能很有用，因为它们通常忽略系统内部。最后，对于临时和长时间运行的工作负载，自适应方法是最佳选择。

2.2 Apache Spark

Apache Spark 是最流行的大数据批处理分析平台，全球 500 强企业中有 80%使用 Apache Spark 处理数据，它是一个开源的分布式框架，它以容错的方式简化了在计算集群上并行执行的应用程序的开发。它的接口基于弹性分布式数据集（Resilient Distributed Dataset，简称 RDD）的概念，RDD 是一组具有只读属性的对象，分布在集群上，并以容错方式维护[38]。作为一种编程抽象，RDD 表示可以通过高性能计算集群拆分的多组对象。对 RDD 的操作也在跨集群并行，从而实现快速和可扩展的并行处理。整体架构由 Spark Core 和一组库 Spark SQL、 Spark MLlib，、Spark GraphX 和 Spark Streaming[39]。Spark 系统内包含的配置参数数量有 200 个左右，这些参数主要影响计算资源的执行和分配，包括 CPU、内存、磁盘带宽和网络利用率[40]。

2.3 研究动机

虽然确定 Spark SQL 应用程序配置参数的值域通常需要一定的时间成本，但其作用至关重要，这是由于 Spark SQL 应用程序通常在长时间内会被重复执行多次。在一次机器学习调优过程中，收集样本的数量通常为 1000 到 10000 个，每个程序执行的时间为几分钟，如果配置参数的值域范围设置不合理，通常会导致错误样本数量为有效样本数量 5 倍，10 倍甚至更多，使时间开销由原来的几个小时变成几天甚至几周，而且对集群资源也造成了极大地浪费。如图 1 所示，在对 Spark SQL 程序进行配置参数调优收集训练样本的过程中，由于配置参数的不合理设置，程序运行的成功率只有 46.5%，其中有些不合理的参数设置还会导致系统长时间挂住甚至崩溃，导致集群故障性关闭或重启，这样会导致时间开销成倍增加，例如 1000GB 数据集大小下的 q72 的配置参数调优的样本收集阶段需要执行 1000 次，每次平均耗时 10 分钟，那么原本需要 10000 分钟（约 160 小时），由于其成功率只有 42%，耗时变为 400 小时，平均增加了 10 天。由此可见，确定 Spark SQL 应用程序的配置参数值域至关重要且具有重大的工业层面的实践意义。

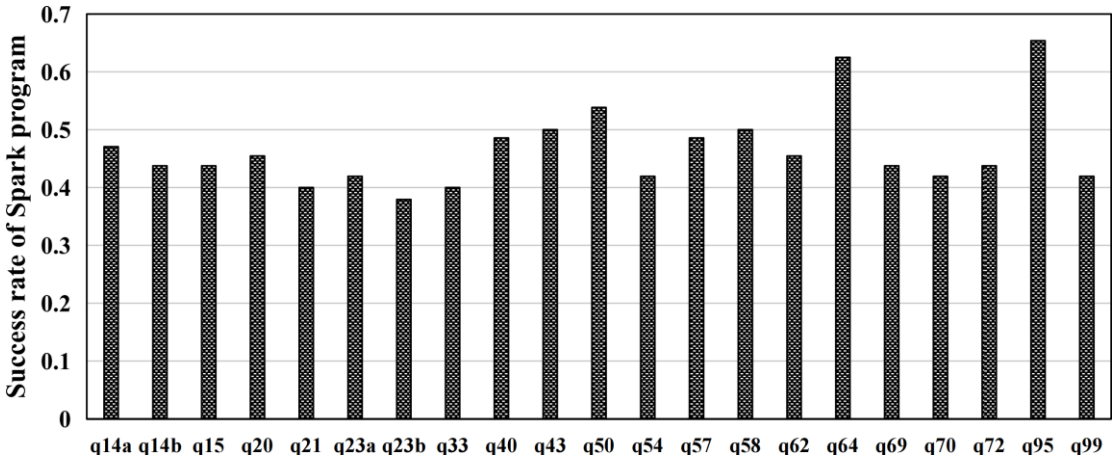

图 1 Spark SQL 程序在机器学习调优过程中程序运行成功率

Fig. 1 The success rate of a Spark SQL program during machine learning tuning

3 方法论

3.1 复合搜索

如何能在左边界是 0，右边界是正无穷大的数轴上将配置参数的搜索区间确定下来呢？首先我们可以确定的是探测的起点，也就是 Spark 官方文档所给出配置参数的 default 值。从初始值出发，很容易就想到的是每一步都以固定的步长，向左右两边逐步探测。当探测到达边界时，系统会给出越界反馈。当左右两边都给出越界反馈时，我们分别取左右两边最后一次未越界的值作为左右边界，这样值域就确定下来了。然而，这种直观的求解过程存在以下几个问题：

首先，固定步长的取值会极大地影响搜索的效率，比如执行器的内存通常取值为 1GB，但集群资源的内存数通常为 10GB 左右，如果我们以 1GB 为步长探测右边界，所得到的右边界值假设为 8GB，虽然需要 8 步就能得到右边界，但是边界的精度只停留在 GB 级别，这对值域的参考价值非常有限。换言之，我们如果采用更细粒度的 100MB 作为步长，那么我们得到右边界的值虽然精确度达到了 100MB 级别，但是需要 80 步才能得出结果，时间成本提高了 10 倍，这是非常难以接受的，很难达到时间与精度的平衡点。因此我们针对该问题提出了一种复合搜索的方法，能够实现在非确定区间和无确定搜索目标值的糟糕情况下，利用 Spark SQL 程序配置参数在越界情况下的几种约束条件，实现配置参数值域的快速探测和确定，我们称之为复合搜索。

根据上述，我们可以很容易根据配置参数的功能属性，推断出数值型配置参数的值域范围是一个连续区间，且存在上下界。除了数值类型的配置参数，还有布尔类型，固定值类型的配置参数，由于其值域显而易见，我们在此不做讨论。关于数值类型的配置参数，又可以划分为连续型和离散型，连续型由于其可以取到的值域范围比离散型稠密复杂得多，因此我们的复合搜索主要研究的对象是连续数值类型配置参数，当这个问题解决时，方法自然可以推广到离散数值类型配置参数，以及布尔型和固定值配置参数。

我们可以将 CS 的关键技术分为两部分：第一部分是搜索区间的初步探测（Preliminary Interval Detection，简称 PDI），这是本方法的核心部分，因为要解决问题棘手的地方就在于区间和目标值的不确定性，PDI 可以通过高效的指数型试探法，对搜索区间进行确定，从而可以利用传统的搜索方法进一步探测；第二部分是区间上的斐波那契搜索（Fibonacci Search，简称 FBS），这一部分主要是在 PDI 初步确定的区间上进一步搜索，最终确定配置参数的上下界，即值域范围，具体求解过程，以求解右边界为例（求解左边界的过程相当于右边界的镜像），大致流程图如图 2 所示：

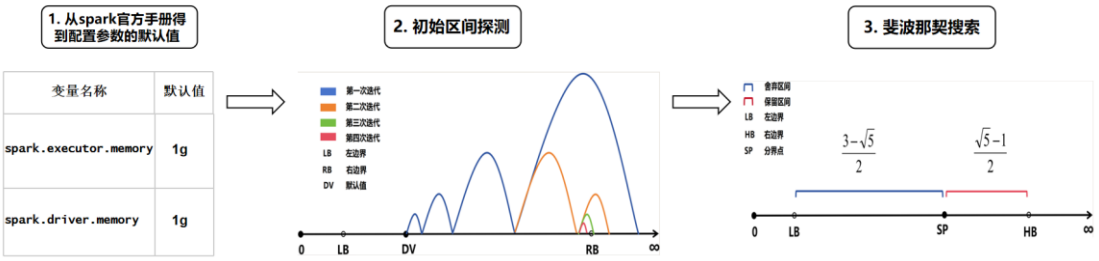

图 2 复合搜索流程图

Fig. 2 Composite search flowchart

3.1.1 越界反馈

在 Spark SQL 程序运行过程中，由于配置参数的不合理设置，会出现以下两种情况：（1）程序执行失败。（2）程序长时间挂住。复合搜索是利用这些异常情况，在实际搜索过程中来回反复利用该反馈条件，从而逐步逼近真实的 Spark SQL 配置参数值域范围。我们

将以上反馈条件出现的情况称为越界反馈。这种思想在复合搜索的第一部分初始区间探测中体现得尤为明显。下面我们举例说明两种越界反馈的区别和应用场景。

对于越界反馈（1），当搜索过程中配置参数 spark.executor.memory 被设置为 50GB，而集群的 memory 资源量只有 40GB 时，Spark SQL 应用程序显然会因为内存越界错误（Out Of Memory，简称 OOM）而执行失败。值得关注的是，当 spark.executor.memory 被设置为 40GB，而集群的 memory 资源量有 50GB 时，同样会发生 OOM 错误，这是由于在实际运行过程中，Spark 的内存分配不止包含 executor 部分，还包含 driver 部分，压缩和序列化的缓冲区（buffer）大小以及集群机器本身操作系统所需的内存部分。因此我们利用显式的报错以及故障，可以在程序实际运行中实时检测，一旦出现该越界反馈，我们就认为配置参数超出了上下界，从而逐步逼近配置参数真实的边界。

对于越界反馈（2），当搜索过程中配置参数 spark.sql.autoBroadcastJoinThreshold 设置过大时，会出现数据量巨大的数据表全部通过广播传播到各个集群节点的情况，这会导致严重的网络拥塞以及巨大的时间开销，程序会长时间挂住而无法正常运行。因此我们利用设置时间阈值的方式，排除错误的配置参数设置，同样可以有效地逼近配置参数的值域范围。

3.1.2 初始区间探测

为了易于数学分析，我们将越界反馈现象抽象成表征 Spark SQL 程序执行的反馈函数 $F(x)$，由于 $F(x)$ 在$(0,\infty)$上连续且在值域范围 (a,b) 上 $F(x)>0$，在 $(0,a)$ 和 (b,∞) 上 $F(x)<0$，根据介值定理，在$(0,\infty)$上必定存在两个根 x_1 和 x_2，使得 $F(x)=0$，这两个根 x_1 和 x_2 即是配置参数的值域分别是值域的左边界 a 和右边界 b。PDI 主要由三个步骤组成（由于左右边界搜索过程可以镜像推演，因此以搜索右边界 b 为例说明）：

（1）配置参数 CP 由官方文档提供的默认值 D 出发，探测右边界 b，n 表示当前迭代过程中的总步数，并且初始 $n=1$，每一次向右探测的步长 S 取值为 $D\times 2^{n-1}$。

（2）如果当 $\mathrm{n}=\mathrm{k}$ 时出现越界反馈，则令 CP 取步数为 $n-1$ 时的值，并且更新步数 $n=1$，步长 S 为 $D\times 2^{k-n}$。

（3）重复步骤（2），直至反馈函数 $F(x)=0$。

当 CP 首次出现越界反馈，步长 S 的变化趋势由原来的指数型增长变为指数型下降，这是由于我们既期望快速达到边界点附近——因此采用了指数型增长步长来初步试探越界反馈的边界，又期望能充分利用越界反馈条件迅速收敛到边界点——因此采用了指数型下降来快速收敛到边界点。如图 3 所示，我们采用变量 L_k 表示第 k 次迭代后时，右边界 b 所在的区间长度，则

$$L_k = D+\sum_{n} S-(D+\sum_{n-1} S)=\sum_{n} S-\sum_{n-1} S=2^{n-1} \quad (2)$$

其中 n 表示当前迭代过程中的总步数，由于初始区间长度 L_0 是未知的，但经过第一次迭代后 L_1 代替了 L_0 作为区间长度的收敛起点，因此可得递推关系

$$L_k = \frac{L_1}{2^{\sum_{k} n_1+n_2+\ldots.+n_k}} \quad (3)$$

当 $k\to\infty$ 时，$L_k\to 0$，即区间长度趋近于零，由上述介值定理，在区间 $[D,\infty]$ 上必定存在一个根 x_b 使得 $F(x)=0$。随着迭代的进行，区间不断缩小，并且根 x_b 必定收敛到右

边界 b 。

对于 **PDI** 的收敛速率，由于每次迭代区间收缩的比例视越界反馈所需步数不同，但均为 2^n，即指数次幂，因此可以表示为 $F_k \sim 2^n$，时间复杂度可表示为 $\mathrm{O}(\log n)$ 。

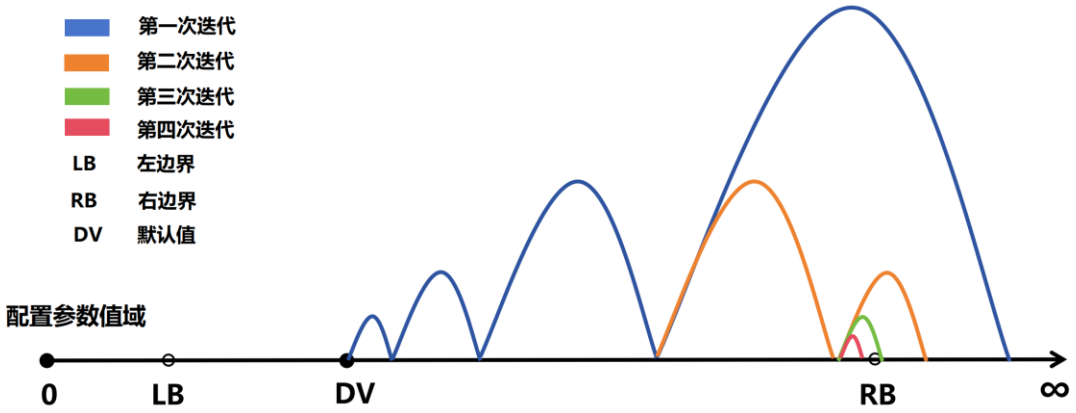

图 3 初始区间探测（PDI）的具体实现过程

Fig. 3 The specific implementation process of PDI

3.1.3 斐波那契搜索

斐波那契搜索是基于斐波那契数列逐步缩小区间的策略，如图 4 所示。斐波那契搜索与二分搜索类似，我们通过对比来分析斐波那契搜索的特性，它们的主要区别如下：

（1）区间更新方式：二分搜索是每次计算区间中点 $m = \dfrac{a+b}{2}$，并评估函数在中点处的值。根据单峰函数的性质，选择其中一个子区间继续搜索；而斐波那契搜索是基于斐波那契数列逐步缩小区间的策略。在每次迭代中，斐波那契数列的比例用于选择搜索区间的两个点 $x_1 = a + \dfrac{F_{k-2}}{F_k}(b-a)$ 和 $x_2 = a + \dfrac{F_{k-1}}{F_k}(b-a)$，然后根据函数值决定如何更新搜索区间。这里黄金分割数可以近似取 0.618，分数表示形式如图 4 所示。

（2）收敛速度：二分搜索在二分搜索在每次迭代中将区间长度减半，收敛速度是对数级别的，即每次迭代使区间长度缩小一半。对于 n 个步骤，二分搜索最多需要 $\mathrm{O}(\log n)$ 次迭代；而斐波那契搜索的收敛速度比二分搜索更快，因为斐波那契数列的增长速度比等比数列的增长速度快。具体地，斐波那契搜索的收敛速度是指数级的，通常需要 $\mathrm{O}(\log n)$ 次迭代，但由于使用了斐波那契数列，区间收缩的方式更为平滑且高效。

（3）适用场景：二分搜索广泛应用于已经排序的单峰函数或者单调递增或递减的函数。二分搜索的优势在于其简单性和高效性，但其只适用于函数是单峰的或单调的情况下，并且每次需要计算函数值的中点位置；而斐波那契搜索在单峰函数上也非常有效，尤其是当无法计算中点或不希望每次都计算中点时。它可以减少对中点的依赖，适用于一些不容易计算中点的场景。

（4）时间复杂度：二分搜索的时间复杂度为 $\mathrm{O}(\log n)$，每次操作通过将搜索区间的一半排除掉；斐波那契搜索的时间复杂度也是 $\mathrm{O}(\log n)$，但由于其区间缩小的方式依赖于斐波那契数列，因此在常数因子上略高于二分搜索。

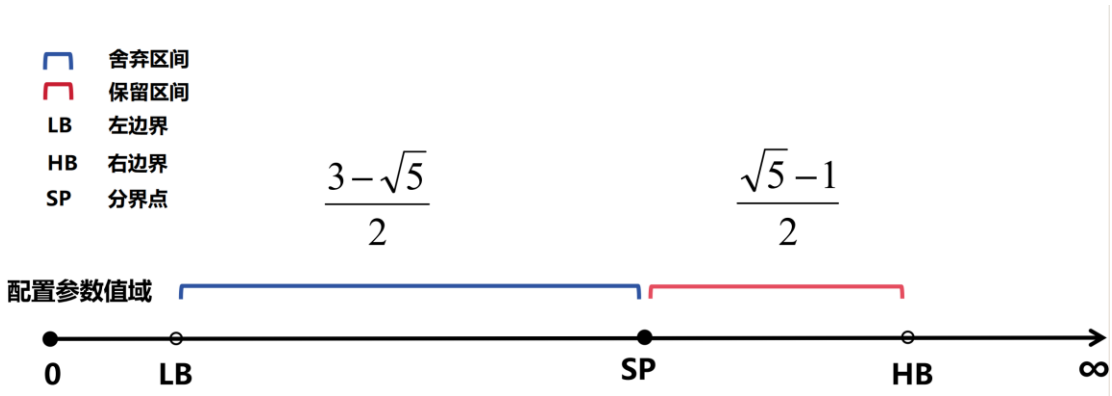

图 4 斐波那契搜索（FBS）的具体实现过程

Fig. 4 The implementation process of FBS

3.2 依赖关系分析

由于配置参数之间存在错综复杂的依赖关系，某些参数的取值往往会对其他参数的值域范围有影响。比如最简单的一个例子，*spark.memory.offHeap.enabled* 决定了是否使用除了分配给 jvm 内存以外的堆外内存。而 *spark.memory.offHeap.size* 决定了分配给 jvm 堆外内存的大小，如果前者的值为 False，那么后者的值大小就没有任何实际意义[41]。此外，cores 相关与 shuffle 相关的参数依赖关系要复杂的多，cores 相关的还能够用乘积等式表示，而 shuffle 相关的参数在执行期间的各个阶段发挥不同的阶段，例如物理计划中的 join 操作，与 *spark.sql.autoBroadcastJoinThreshold*，*spark.reducer.maxSizeInFlight*，*spark.sql.join.preferSortMergeJoin* 等参数相关，*aggregate* 操作与 *spark.sql.codegen.aggregate.map.twolevel.enable* 和 *spark.shuffle.io.numConnectionsPerPeer* 相关等等，这些依赖关系往往是隐式的，很难用具体的等式或者大小关系来表征。

因此我们将联系比较密切的相关参数分类进行处理，例如 memory 相关的配置参数，可以将每个 worker 节点分配给 executor 的执行内存 spark.executor.memory 与 executor 的数量 spark.executor.instances 相乘，然后与堆外内存以及额外内存相加，即可得到分配给 Spark SQL 应用程序的总内存。对于网络传输相关的配置参数，我们可以在类内分成两小类，首先是通过广播，缓冲或者压缩的方式，平衡内存与网络开销。这些相关的参数包括 spark.sql.autoBroadcastJoinThreshold，spark.shuffle.file.buffer 等。另一类是通过延时或者设定阈值的方式，平衡网络带宽和延迟，包括 spark.locality.wait，spark.reducer.maxSizeInFlight，这两种小类是相互协调制衡的。当网络带宽资源充足，内存资源有限时，可以适当增大前者类内配置参数的值；当内存资源充足，而网络拥塞时，可以调节后者类内配置参数的值，从而协调网络和内存资源的利用。下面依次介绍内存相关，Shuffle 相关和网络带宽相关的配置参数之间的依赖关系。

3.2.1 内存相关

由于 Spark 的内存计算特性，找到好的内存参数设置对性能至关重要。内存消耗主要包括两种：执行和存储。执行中的内存是指 shuffle、join、排序和聚合阶段的内存消耗，而存储中的内存是指用于缓存和跨集群传播输入或中间数据的内存消耗。这两个类别都被设计为共享一个统一的区域 M，这意味着当不使用执行内存时，存储可以获取所有可用内存，反之亦然。在执行过程中，内存可能会在需要时驱逐存储，但只有在仍然有一定阈值

R 可用的存储内存时才会这样。R 指定统一区域 M 中的子区域，在该子区域中缓存的块永远不会被驱逐[42]。两个最相关的内存配置是:（1）spark.memory.fraction，它表示用于 M 的内存比例，默认设置为 0.6。其余的用于数据结构、元数据和保护;（2）spark.memory。storageFraction，它表示阈值大小 R 默认设置为 0.5。

3.2.2 Shuffle 相关

Shuffle 是代价很高的操作，因为 Shuffle 数据必须被写入磁盘，然后通过网络传输。因此，避免常见的陷阱并选择正确的排列可以显著减少 shuffle 次数并提高应用程序的性能。重分区、共组、连接以及所有的*By 或*ByKey 转换都可能导致 shuffle 操作[43]。应该考虑以适当的方式使用这些转换:(a) groupByKey 执行关联约简操作，该操作在网络上传输整个数据集;(b) reduceByKey 按键聚合值，计算每个分区中每个键的聚合值;(c)应避免使用 flatMap-join-groupBy——当两个数据集涉及到一个 groupByKey 时，最好使用 cogroup，因为组需要拆封和重新打包[44,45]。

3.2.3 网络带宽相关

Spark 应用程序中的网络带宽利用率和许多个配置参数密切相关，其中有利用内存存储来减少网络开销的配置参数，如 spark.sql.autoBroadcastJoin-Threshold，它通过控制广播表的阈值大小，从而控制广播到各个节点表的大小，但在 Spark SQL 程序执行的过程中，如果用到表中的数据，就需要利用网络传输到各个节点，频繁的数据获取会造成严重的网络拥塞问题[46]。但如果增大广播表的阈值大小，一开始就把各个表传播到各个节点，利用内存存储，这样在程序执行过程中就可以在本节点获取数据而不需要进行网络传输。另外还有利用时延来解决网络拥塞问题的 spark.locality.wait，它表示程序在排队运行时，优先选择利用本地的资源，如首先考虑其他 RDD 分区有没有空闲，如果有则程序准备就绪，开始执行，如果没有空闲，则等待 spark.locality.wait 设置的时间间隔 t 后，开始向上一级 executor 寻求资源分配，如果 executor 仍然没有空闲，则需要进一步等待时间间隔 t，向 executor 上一级 node 寻求资源分配[47]。这样的时延措施，可以有效缓解网络拥塞问题。

4 实验结果及分析

4.1 复合搜索的性能加速比

为了验证复合搜索和建立分析模型相结合方法的有效性，我们的实验主要是在 6 台 X86 架构的服务器上，对大数据基准测试集 TPC-DS 的 23 个 Query 进行了分析。分别采用传统的固定步长探测法与复合搜索法对 Spark SQL 应用程序，在 500GB 大小数据集上进行值域范围的求解。其中，固定步长探测我们在处理不同参数时，所选取的固定步长增量不同，如针对 spark.executor.memory，我们选取的步长增量为 1GB；而针对 spark.sql.auto-BroadcastJoinThreshold，我们选取的步长增量为 1MB。这种差异是由参数本身的属性决定的，类似的我们在使用复合搜索法时，也会针对不同的参数设计不同的值域范围的搜索方案。以下是我们在不同维度对复合搜索和依赖关系分析相结合，其中包括 TPC-DS 的对配置参数调节表现敏感的 Query 维度及对 Spark 性能表现重要的配置参数维度，主要涉及 10 个具有代表性的 query 分类如表 1 所示，其中迭代计算型指的是在数据的处理包含迭代过程，例如首先确定连续两年在所有三个销售渠道销售的相同品牌、类别和类别的商品；查

询分组型指的是在计算过程后有分组排序等操作，例如计算指定项目类别和时间段的总收入以及总收入与收入的比率（按项目类别）；计算密集型指的是计算过程复杂且涉及表较多，例如计算通过目录渠道销售的促销商品的平均数量、标价、折扣、销售价格；计算存储型指的是在计算过程后有分组存储等操作，例如计算通过商店销售的总数量、退回的数量和通过目录购买的数量。将这些信息按项目分组并存储。

表 1 TPC-DS 具有代表性的十个 Query 的分类

Table 2 Categories of the ten representative queries in TPC-DS

Query 类型	Query 名
迭代计算型	q14a,q14b,q23a,q23b
查询分组型	q15,q21,q33
计算密集型	q20
计算存储型	q29,q40

4.1.1 程序维度分析

由图 5 和图 6 所示，我们可以看出复合搜索相比传统方法的性能提升是非常明显的，我们所列出的 10 个对配置参数敏感的 query 的性能加速比平均为 6.5 倍，对于 q23a 来说，最大为 17.6 倍的提升，对于 q15 最少也有 1.5 倍提升，这是由于 q15 运行时间比较短，约为 1 分钟，SQL 语句的结构很简单，只针对数据量较小的 catalog_sales 这一个数据表进行 shuffle 操作，因此 q15 配置参数的值域比较规范，使用传统方法就能比较高效的搜索，而使用复合搜索带来的增益会不明显。但对于 q23a 来说，SQL 语句非常复杂，涉及的数据表高达 12 个，其中有 4 张较大的数据表和 8 张维度表，因此参数之间的依赖关系复杂，并且真实值域往往也会难以探测，因此我们使用高效的复合搜索对性能的提升是非常显著的。

4.1.1 配置参数维度分析

对于每一个配置参数而言，其各自在不同的 Spark 应用程序里发挥的作用不同，表 2 是分析了 Spark SQL 应用程序的配置参数在 TPC-DS 的 103 个 query 里的重要性因子，并对数据进行加和求平均所得到的每个重要配置参数的权重系数，我们发现，排名前 10 的配置参数，有 7 个资源密切相关配置参数，2 个并行度相关配置参数和 1 个广播相关配置参数。我们可以看到 spark.sql.shuffle.partitions，spark.sql.autoBroadcastJoin-Threshold 这两个配置参数所占权重是非常大的，因此我们重点考虑。另外我们选取了四个与资源密切相关的配置，其中有 spark.executor.instances，spark.driver.cores，spark.driver.memory，spark.executor.-memory 等，还有一个并行度相关的 spark.default.-parallelism，分别使用复合搜索和传统方法，对其值域进行求解。结果表明，复合搜索相对传统方法性能加速比平均为 5.9 倍，其中加速比最大的配置参数为 spark.sql.-autoBroadcastJoinThreshold（BJT），为 15.7 倍，这是由于 BJT 本身是一个数值非常大的配置参数，单位选取为 KB，默认值为 10240，复合搜索对这种细粒度的变量来说，性能提升的效果更加显著，而对于一些值域范围比较狭窄的配置参数，比如 spark.executor.-cores，通常数量为 1-10 个，性能提升效果往往会比较差，甚至几乎没有提升。

表 2 **Spark SQL 应用程序重要性排序前十的配置参数**

Table 2 Top 10 configuration parameters for Spark SQL program importance

参数名	权重
spark.sql.shuffle.partitions	18.75%
spark.sql.autoBroadcastJoinThreshold	16.82%
spark.executor.instances	5.84%
spark.executor.cores	4.75%
spark.default.parallelism	4.74%
spark.driver.cores	4.56%
spark.executor.memory	3.61%
spark.memory.offHeap.size	3.41%
spark.memory.storageFraction	2.95%
spark.driver.memory	2.88%

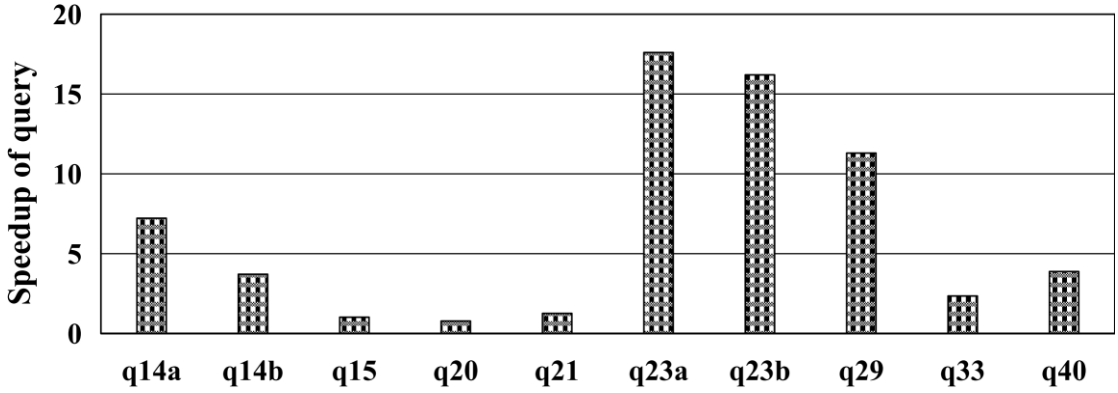

图 5 Query 维度上复合搜索相对传统固定步长搜索的加速比

Fig. 5 The speedup ratio of compound search compared with fixed-step search in the Query dimension

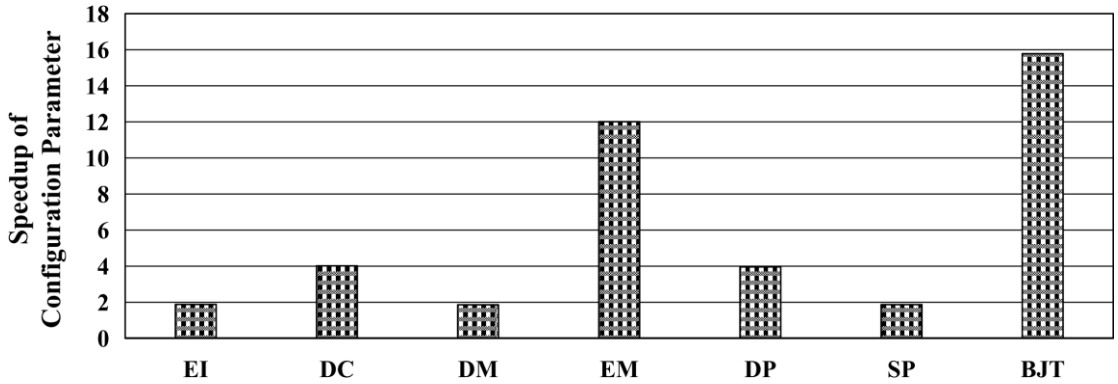

图 6 配置参数维度上复合搜索相对传统固定步长搜索的加速比

Fig. 6 The speedup ratio of compound search compared with fixed-step search in the CP dimension

EI：spark.executor.instances **DC: spark.driver.cores**

DM: spark.driver.memory **EM: spark.executor.memory**

DP: spark.default.parallism **SP: spark.shuffle.partitions**

BJT: spark.sql.autoBroadcastJoinThreshold

4.2 Spark 程序运行成功率提升

复合搜索相对传统搜索方法的速度提升是非常明显的，接下来我们将搜索确定下来的 Spark SQL 配置参数的值域应用在实际集群上，通过 Spark SQL 程序运行成功率来观察值域范围的有效性。程序运行的成功率是指结果如图 7 所示。实验是在在相同的 Spark SQL 应用程序的机器学习调优过程中，使用复合搜索所得值域范围进行训练样本的收集，和传统方法进行对比。结果表明，程序运行的成功率由原来的 46.5%提升到了 81.7%，其中最大为 q43 的 95.2%，最小为 q23b 的 74.1%。实验结果进一步说明了我们使用复合搜索得出的值域范围的精确度要远高于传统方法，结合 4.1 节中关于复合搜索相对传统方法加速比的提升，我们可以得出结论：复合搜索相对于传统方法，无论是在时间开销还是在精确度上均有显著提升，这也证明了我们使用复合搜索能够实现快速精确确定 Spark SQL 配置参数值域。

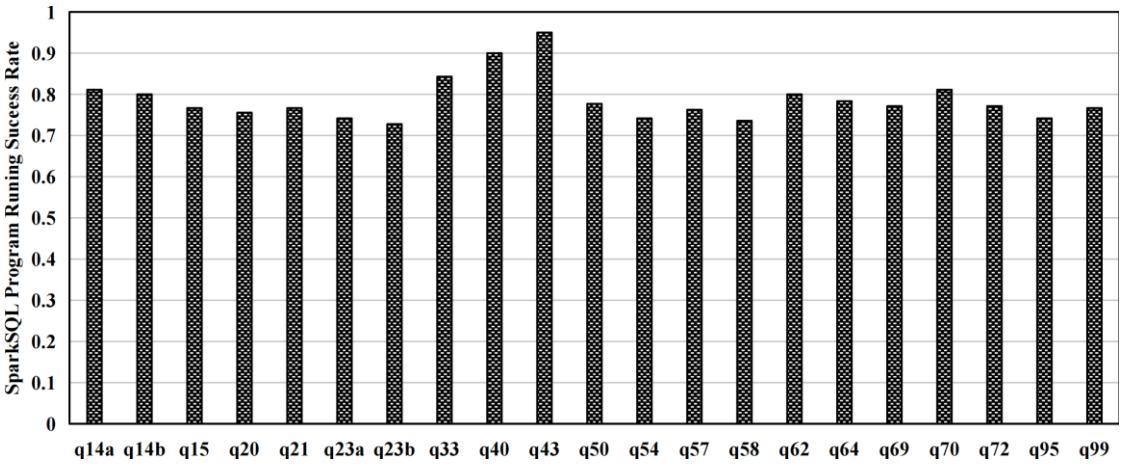

图 7 复合搜索值域下 Spark 程序运行的成功率

Fig. 7 The success rate of the Spark program in the compound search value domain

4.3 Spark 程序调优时间开销显著减少

复合搜索作为一种快速确定 Spark 应用配置参数值域的方法，可以广泛与其他 Spark 配置参数调优的方法相结合，从而有效地减少 Spark 程序调优的时间开销。通过几组对比实验，如图 8 所示，可以观察到在应用了复合搜索确定值域范围的前提下，几种 Spark 调优方法在 TPC-DS 的 100GB 数据集大小下 6 个执行时间最长的 query（q14a，q14b，q23a，q23b，q64，q72）调优时间开销的对比。其中，我们主要选取了具有代表性的实验驱动调优方法 DAC，机器学习调优方法 LOCAT 和 QTune[39]。图中 ADDCS 表示应用了复合搜索之后 Spark 程序调优的时间开销（其中包含复合搜索确定值域范围的时间开销以及在值域范围下调优的时间开销），BASE 表示原方法调优时间开销，纵轴时间单位是 h。我们可以观察到，在使用复合搜索后，无论是实验驱动调优还是机器学习调优，性能均有提升。其中，对于 DAC 来说，时间开销平均减少 30%，最高减少 52.8%；对于 LOCAT 来说，时间开销平均减少 28.1%，最高减少 46.9%；对于 QTune 来说，时间开销平均减少 27.4%，最高减少 44.5%。综合来看，复合搜索对于现有的机器学习调优和实验驱动的调优方法的性能提升作

用很大，能够有效降低因配置参数设置不当造成的不必要的时间成本。

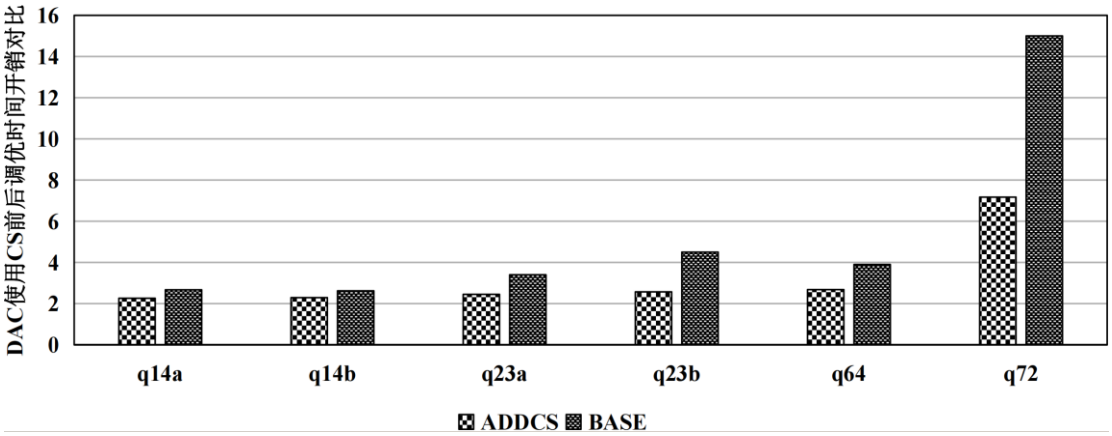

图 8 DAC 在使用复合搜索前后对 6 个 query 调优时间开销对比

Fig. 8 DAC compares the tuning time and overhead of six queries before and after using composite search

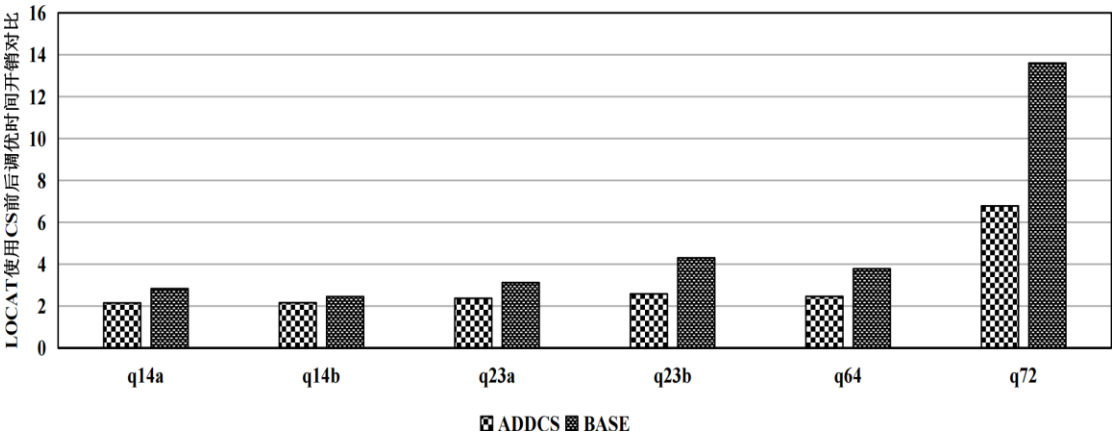

图 9 LOCAT 在使用复合搜索前后对 6 个 query 调优时间开销对比

Fig. 9 LOCAT compares the tuning time and overhead of six queries before and after using composite search

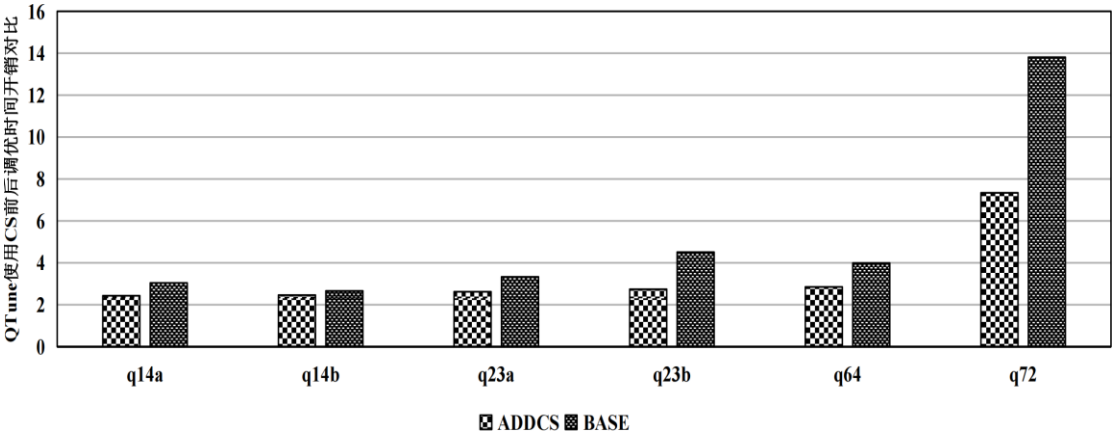

图 10 QTune 在使用复合搜索前后对 6 个 query 调优时间开销对比

Fig. 9 QTune compares the tuning time and overhead of six queries before and after using composite

4.4 Spark SQL 配置参数的推荐值域

我们分析了复合搜索在求解 Spark SQL 配置参数值域上的优异表现，接下来我们根据 TPC-DS 不同 Query 的运行时间以及 SQL 程序的复杂程度，确定了实验环境集群的推荐配置参数值域。首先我们将每个 Query 的平均运行时间 T，SQL 语句的长度 L 和 DAG 图的复杂程度 D 等作为特征，生成推荐值域中 TPC-DS 的 103 个 query 各自的权重。例如 q72 的运行时间在所有的 query 里最长，因此在推荐值域中，q72 利用符合搜索确定值域的上下界在最终结果中的权重为 7.78%。其他诸如 q23b，q80 等等都是 SQL 语句结构比较复杂的 query，它们的权重也相应分别为 4.38%和 4.17%，对最终结果的影响较大。表 2 中给出的是前 7 个对推荐值域重要的 query 所占的权重，表 3 是我们根据权重求得的最终对于集群和大部分 Spark 程序在实际应用中匹配的配置参数的值域范围。

表 2 推荐值域中 7 个权重占比 TPC-DS 的 query

Table 2 The query of the TPC-DS in which several weights in the recommended value range

Query 名称	权重
q72	7.78%
q23b	4.38%
q80	4.17%
q64	3.95%
q23a	3.95%
q29	2.08%
q17	2.08%

5 分析与讨论

现有的参数调优方法解决了高吞吐量和资源利用率、快速响应时间和成本效益方面的各种挑战。针对不同的挑战和场景，提出了不同的策略或方法。这些方法可分为以下六类，我们分别与本文的复合搜索方法做了对比：

基于规则的方法依赖于人类专家的经验和调优说明，帮助用户快速对系统进行参数调整。这种方法通常不需要复杂的模型或日志信息，适合于在没有大量数据和计算资源的情况下快速获得配置建议。通过一套固定的规则，用户可以在不深入了解系统工作机制的前提下进行调优，通常这种方式能够简化配置过程[17,18,19]。优点在于它的操作简便且快速，不需要对系统进行深度分析或监控，可以在短时间内完成调整。与此同时，它的缺点也十分明显，最主要的是缺乏灵活性。在面对动态变化的环境时，基于规则的方法往往难以适应复杂的系统需求。因为它依赖于事先设定的规则，因此当系统出现新的需求或不确定因素时，规则可能不再适用，无法提供足够的应对能力。而复合搜索可以适应动态变化的环境，在不同的集群资源配置下，能够找出相应的配置参数值域，进一步降低调优的时间开销。

成本建模方法是一种通过对系统内部结构和工作机制进行深入分析，构建高效的性能预测模型的调优方式。这种方法主要依赖于白盒模型，通过建立成本函数来预测和优化系

统性能[20,21,22]。其优势在于能够提供较为准确的性能预测，尤其是在拥有足够实验数据和对系统有深刻理解的情况下。利用成本建模，开发者可以精确分析各项参数对性能的影响，并在此基础上进行优化调整。这对于需要高效资源调度和性能预测的系统尤为重要，例如分布式计算和大规模数据处理等。然而，这种方法也存在一定的挑战，首先是构建成本模型所需的实验数据量较大，且要求开发者必须对系统有深刻的理解，尤其是在复杂系统中，建立准确的成本模型是一项非常复杂且耗时的任务。而复合搜索可以采用少量的实验数据，就可以确定在当前集群下的配置参数范围，并且能够运用到构建成本模型的过程中，提高样本数据的质量以及降低收集样本数据时间开销。

基于仿真的方法通过建立虚拟模型对系统进行模拟，以评估不同配置对系统性能的影响。通过模拟，用户可以在不实际改变生产环境的情况下，预估不同参数设置对系统的效果[23,24,25]。这种方法特别适用于需要对大量参数进行试探性调整的场景，例如在数据中心资源调度和网络拥堵预测等应用中，仿真能够提供直观的反馈并帮助决策者做出优化选择。它的优势在于高度的灵活性和安全性，用户可以在仿真环境中调整配置，而无需担心会影响实际系统的稳定性。同时，仿真方法也能够帮助开发者直观地理解和展示不同配置对系统性能的具体影响，帮助快速定位瓶颈并进行优化。然而，仿真方法也存在计算开销较大和仿真精度可能不足的问题。为了模拟真实系统的行为，仿真模型通常需要复杂的计算，并且仿真结果未必完全反映实际系统的性能，因此存在一定的不确定性。而复合搜索具有严谨的数学推导以及扎实的实验数据支撑，对实际系统中的配置参数优化问题进行了改进，因此更具备实践优势。

实验驱动方法依赖于实际的运行数据和反馈，通过反复进行实验来调优系统参数。这种方法通常在没有详细理论模型的情况下使用，依靠反复实验和数据分析来逐步优化系统性能。用户通过改变系统的参数设置，观察运行结果，并根据反馈调整策略，最终找到最佳配置[26,27,28,29,30]。实验驱动方法的最大优势在于它能够提供与实际系统环境高度一致的调优效果。通过实验的实时反馈，用户可以迅速发现潜在的性能瓶颈并进行调整，从而达到优化目标。然而，实验驱动方法也存在显著的缺点，最突出的是它的时间和资源消耗。在某些系统中，反复进行大量的实验可能需要很长的时间和大量的计算资源。此外，实验的设计和分析也需要仔细考虑，确保实验结果具有代表性和可靠性，否则可能导致不准确的结论，甚至错误的调优决策。而复合搜索则只需要少量的时间开销和实际集群的运行环境，即可推导出配置参数的值域范围，从而改进配置参数优化的方案。

机器学习方法通过数据驱动的方式来构建系统的性能预测模型。这种方法通常不依赖于系统的内部机制，而是通过分析大量历史数据来发现影响性能的关键因素。机器学习方法的一个重要优势是它能够自动从数据中学习系统的行为，并在此基础上进行优化。尤其是在面对复杂系统时，机器学习方法能够处理大量高维数据，自动识别潜在的规律，从而进行优化决策。此外，机器学习方法不需要开发者对系统内部有过多了解，系统本身被视为一个“黑盒”，通过大量数据进行训练，模型能够随着数据的变化不断自我调整[30,31,32,33,34,35]。尽管如此，机器学习方法也有其局限性，首先它依赖于大量的高质量数据，这对于某些应用场景来说可能是一个挑战。其次，训练机器学习模型需要较高的计算资源，并且训练过程较为复杂。在数据不足或计算资源有限的情况下，机器学习方法的效果可能会受到影响。而复合搜索恰恰可以为机器学习调优方法提供便利，首先通过少量实验确定配置参数的值域范围，然后在至于范围内收集高质量数据，建立更具鲁棒性的机器学习模型，从而提高了调优的精确度和有效降低了时间开销。

自适应方法是指在应用程序运行过程中，实时监控系统状态并根据环境变化自动调整配置参数。这种方法适用于长期运行且对实时响应有高要求的应用，能够在不断变化的环境中保持较为稳定的性能。自适应方法通过实时监控系统负载、资源使用等指标，根据不

同的工作负载和环境变化动态调整参数，以确保系统始终能够高效运行。其主要优点在于能够自适应调整系统配置，特别适合于那些长期运行的应用或对响应时间有较高要求的系统。自适应方法能够通过动态调整应对变化的负载和环境，从而避免系统因过载或资源分配不均导致的性能下降。尽管如此，自适应方法的实施相对复杂，需要建立实时监控和反馈机制，并对调整策略进行精细化设计。因此，这种方法的部署和维护成本较高。此外，实时反馈和调整也可能带来额外的计算开销，因此在一些资源有限的环境中，可能需要在自适应调优和系统负载之间做出权衡。而复合搜索可以通过确定配置参数的值域，为自适应方法提供初始值以及动态调节的变化范围，从而提高自适应方法的效率，降低其时间开销以及对资源配置的依赖。

6 结论

本研究通过对 Spark SQL 程序运行的原理和机制，结合配置参数本身的属性，提出了一种高效的求解 Spark SQL 配置参数值域的方法——复合搜索，并结合配置参数间的依赖关系，给出了值域区间求解的解决方案。结果表明，无论是从程序维度还是从配置参数维度上，复合搜索对于 Spark SQL 配置参数的值域搜索相对传统方法的性能提升显著，平均分别为 6.5 倍和 5.9 倍。另外，使用复合搜索所得值域能够使 Spark SQL 程序运行的成功率由 46.5%提升到 81.7%，节约了大量确定配置参数值域范围的时间成本，并且提高了程序运行的成功率。在现有的实验驱动调优与机器学习调优方法中应用复合搜索，能平均减少 30%的时间开销，这对配置参数调优具有重要的意义。

表 3 Spark SQL 配置参数的推荐值域

Table 3 The recommended value range of Spark SQL configuration parameters

参数名	默认值	值域范围
spark.io.compression.zstd.level	1	1-6
spark.io.compression.zstd.bufferSize	32	16-96
spark.executor.instances	2	1-64
spark.driver.cores	1	1-24
spark.driver.memory	1	1-36
spark.memory.offHeap.enabled	False	True-False
spark.memory.offHeap.size	0	512-20480
spark.executor.memory	1	1-64
spark.reducer.maxSizeInFlight	48	32-128
spark.shuffle.compress	True	True-False
spark.shuffle.file.buffer	32	16-96
spark.shuffle.sort.bypassMergeThreshold	200	64-960
spark.shuffle.spill.compress	True	True-False
spark.broadcast.compress	True	True-False
spark.kryoserializer.buffer.max	64	16-128
spark.kryoserializer.buffer	64	8-128
spark.rdd.compress	False	True-False
spark.memory.fraction	0.6	0.32-0.96
spark.memory.storageFraction	0.5	0.08-0.96
spark.broadcast.blockSize	4	1-4
spark.executor.cores	1	4-16
spark.storage.memoryMapThreshold	2	1-16
spark.locality.wait	3	1-16
spark.scheduler.revive.interval	1	1-1024
spark.executor.memoryOverhead	384	384-20480
spark.default.parallelism	CPU cores	1-1024
spark.shuffle.io.numConnectionsPerPeer	1	1-5
spark.sql.shuffle.partitions	200	1-1024
spark.sql.inMemoryColumnarStorage.compressed	True	True-False
spark.sql.inMemoryColumnarStorage.batchSize	10000	8000-12000
spark.sql.inMemoryColumnarStorage.partitionPruning	True	True-False
spark.sql.join.preferSortMergeJoin	True	True-False
spark.sql.sort.enableRadixSort	True	True-False
spark.sql.retainGroupColumns	True	True-False
spark.sql.codegen.maxFields	100	84-128
spark.sql.codegen.aggregate.map.twolevel.enable	True	True-False
spark.sql.cartesianProductExec.buffer.in.memory.threshold	4096	1024-4096
spark.sql.autoBroadcastJoinThreshold	10240	2048-409600

参考文献

[1] Shu Wang, Henry Hoffmann, and Shan Lu. 2022. AgileCtrl: A Self-Adaptive Framework for Configuration Tuning. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE ’22), November 14–18, 2022, Singapore, Singapore. ACM, New York, NY, USA, 13 pages.

[2] Owolabi Adekoya, Habib Sabiu, Derek Eager, Winfried Grassmann, and Dwight Makaroff. 2018. A case study of spark resource configuration and management for image processing applications. In Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering (CASCON '18). IBM Corp., USA, 18–29.

[3] Zhibin Yu, Zhendong Bei, and Xuehai Qian. 2018. Datasize-Aware High Dimensional Configurations Auto-Tuning of In-Memory Cluster Computing. In Proceedings of 2018 Architectural Support for Programming Languages and Operating Systems (ASPLOS’18). ACM, New York, NY, USA, 14 pages.

[4] Jinhan Xin, Kai Hwang, and Zhibin Yu. 2022. LOCAT: Low-Overhead Online Configuration Auto-Tuning of Spark SQL Applications. In Proceedings of the 2022 International Conference on Management of Data (SIGMOD ’22), June 12–17, 2022, Philadelphia, PA, USA. ACM, New York, NY, USA, 11 pages.

[5] Zhenyan Lu, Chao Chen, Jinhan Xin, and Zhibin Yu. 2020. On the Auto Tuning of Elastic-search based on Machine Learning. In 2020 International Conference on Control, Robotics and Intelligent System (CCRIS 2020), October 27–29, 2020, Xiamen, China. ACM, New York, NY, USA, 7 pages.

[6] Chi Li, Shu Wang, Henry Hoffmann, and Shan Lu. 2020. Statically Inferring Performance Properties of Software Configurations. In Fifteenth European Conference on Computer Systems (EuroSys ’20), April 27–30, 2020, Heraklion, Greece. ACM, New York, NY, USA, 16 pages.

[7] Gao Han, Hanyang Shao, Ruiqin Duan, Changqi Zhuang, and Qiao Xiang. 2024. ConfigHub: A Network Configuration Sharing Platform. In Proceedings of the 2024 SIGCOMM Workshop on Formal Methods Aided Network Operation (FMANO '24). Association for Computing Machinery, New York, NY, USA, 33–38.

[8] Runxiang Cheng, Lingming Zhang, Darko Marinov, and Tianyin Xu. 2021. Test-Case Prioritization for Configuration Testing. In Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA ’21), July 11–17, 2021, Virtual, Denmark. ACM, New York, NY, USA, 14 pages.

[9] Shu Wang, Chi Li, William Sentosa, Henry Hoffmann, Shan Lu, and Achmad Imam Kistijantoro. 2018. Understanding and Auto Adjusting Performance-Sensitive Configurations. In Proceedings of 2018 Architectural Support for Programming Languages and Operating Systems (ASPLOS’18). ACM, New York, NY, USA, 15 pages.

[10] Qingrong Chen, Teng Wang, Owolabi Legunsen, Shanshan Li, and Tianyin Xu. 2020. Understanding and Discovering Software Configuration Dependencies in Cloud and Data center Systems. In Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE ’20), November 8–13, 2020, Virtual Event, USA. ACM, New York, NY, USA, 13 pages.

[11] Barroso LA, Hölzle U, Ranganathan P. The Data center as a Computer: Designing

Warehouse-scale Machines. 3rd ed. Cham: Springer, 2019.

[12] Sehgal P, Tarasov V, Zadok E. Evaluating performance and energy in file system server workloads. In: Proc. of the 8th USENIX Conf. on File and Storage Technologies. San Jose: USENIX Association, 2010. 253–266.

[13] Zhang YL, He HC, Legunsen O, Li SS, Dong W, Xu TY. An evolutionary study of configuration design and implementation in cloud systems. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. Madrid: IEEE, 2021. 175–176

[14] Zhang JL, Piskac R, Zhai EN, Xu TY. Static detection of silent misconfigurations with deep interaction analysis. Proc. of the ACM on Programming Languages, 2021, 5: 140.

[15] Mehta S, Bhagwan R, Kumar R, Bansal C, Maddila C: Preventing bugs and misconfiguration in large services using correlated change analysis. In: Proc. of the 17th USENIX Symp. on Networked Systems Design and Implementation. Santa Clara: USENIX Association, 2020. 435–448.

[16] Xiang CC, Huang HC, Yoo A, Zhou YY, Pasupathy S. PracExtractor: Extracting configuration good practices from manuals to detect server misconfigurations. In: Proc. of the 2020 USENIX Annual Technical Conf. USENIX Association, 2020. 18.

[17] D. Ardagna, E. Barbierato, E. Gianniti, M. Gribaudo, T. B.M. Pinto, A. P.C. da Silva, and J. M. Almeida. 2021. Predicting the performance of big data applications on the cloud. Journal of Supercomputing 77, 2 (2021), 1321–1353.

[18] Liang Bao, Xin Liu, and Weizhao Chen. 2018. Learning-based Automatic Parameter Tuning for Big Data Analytics Frameworks. In 2018 IEEE International Conference on Big Data (Big Data). 181–190. https://doi.org/10.1109/BigData. 2018.8622018.

[19] Herodotos Herodotou, Yuxing Chen, and Jiaheng Lu. 2020. A Survey on Automatic Parameter Tuning for Big Data Processing Systems. ACM Comput. Surv. 53, 2,Article 43 (apr 2020), 37 pages. https://doi.org/10.1145/3381027

[20] Mayuresh Kunjir and Shivnath Babu. 2020. Black or White? How to Develop an AutoTuner for Memory-Based Analytics. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (Portland, OR, USA) (SIGMOD '20). Association for Computing Machinery, New York, NY, USA, 1667–1683.

[21] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2018. QTune: A QueryAware database tuning system with deep reinforcement learning. Proceedings of the VLDB Endowment 12, 12 (2018), 2118–2130. https://doi.org/10.14778/3352063. 3352129

[22] Yirong Lv, Bin Sun, Qingyi Luo, Jing Wang, Zhibin Yu, and Xuehai Qian. 2018. Counterminer: Mining big performance data from hardware counters. In 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 613–626. https://doi.org/10.1109/MICRO.2018.00056

[23] Qayyum Rida. 2020. A Roadmap Towards Big Data Opportunities, Emerging Issues and Hadoop as a Solution. International Journal of Education and Management Engineering 10, 4 (2020), 8–17. https://doi.org/10.5815/ijeme.2020.04.02

[24] SparkConf 2022. Configuration - Spark 3.2.1 Documentation. Retrieved March 28, 2022 from https://spark.apache.org/docs/latest/configuration.html

[25] Ameen Alkasem, Hongwei Liu, Decheng Zuo, and Basheer Algarash. 2017. Cloudcomputing: a model construct of real-time monitoring for big dataset analytics using apache spark. In Journal of Physics: Conference Series, Vol. 933. IOPPublishing, 012018.

[26] K Anusha and K Usha Rani. 2020. Performance evaluation of Spark SQL for batchprocessing. In Emerging Research in Data Engineering Systems and Computer Communications: Proceedings of CCODE 2019. Springer, 145 – 153

[27] Lorenzo Baldacci and Matteo Golfarelli. 2018. A cost model for SPARK SQL. IEEETransactions on Knowledge and Data Engineering 31, 5 (2018), 819 – 832.

[28] Qian Chen, Kebing Wang, Zhaojuan Bian, Illia Cremer, Gen Xu, and Yejun Guo.2018. Cluster performance simulation for Spark deployment planning, evaluationand optimization. In Simulation and Modeling Methodologies, Technologies andApplications: International Conference, SIMULTECH 2016 Lisbon, Portugal, July29-31, 2016, Revised Selected Papers. Springer, 34 – 51.

[29] Vladimir Kazakovtsev and Sergey Muravyov. 2022. Application of the automatic selection and configuration of clustering algorithms method for the Apache Spark framework. In Proceedings of the 3rd International Conference on Advanced Information Science and System (AISS '21). Association for Computing Machinery, New York, NY, USA, Article 53, 1 – 5.

[30] Yuhao Li and Benjamin C. Lee. 2022. Phronesis: Efficient Performance Modeling for High-dimensional Configuration Tuning. ACM Trans. Archit. Code Optim. 19, 4, Article 56 (December 2022), 26 pages.

[31] Yixin Wu, Xiuqi Huang, Zhongjia Wei, Hang Cheng, Chaohui Xin, Zuzhi Chen, Binbin Chen, Yufei Wu, Hao Wang, Tieying Zhang, Rui Shi, Xiaofeng Gao, Yuming Liang, Pengwei Zhao, and Guihai Chen. 2024. Towards Resource Efficiency: Practical Insights into Large-Scale Spark Workloads at ByteDance. Proc. VLDB Endow. 17, 12 (August 2024), 3759 – 3771.

[32] Yasmine Djebrouni, Isabelly Rocha, Sara Bouchenak, Lydia Chen, Pascal Felber, Vania Marangozova, and Valerio Schiavoni. 2023. Characterizing Distributed Machine Learning Workloads on Apache Spark: (Experimentation and Deployment Paper). In Proceedings of the 24th International Middleware Conference (Middleware '23). Association for Computing Machinery, New York, NY, USA, 151 – 164.

[33] Yang Li, Huaijun Jiang, Yu Shen, Yide Fang, Xiaofeng Yang, Danqing Huang, Xinyi Zhang, Wentao Zhang, Ce Zhang, Peng Chen, and Bin Cui. 2023. Towards General and Efficient Online Tuning for Spark. Proc. VLDB Endow. 16, 12 (August 2023), 3570 – 3583.

[34] Hafiyyan Sayyid Fadhlillah and Rick Rabiser. 2024. Towards a Product Configuration Representation for the Universal Variability Language. In Proceedings of the 28th ACM International Systems and Software Product Line Conference (SPLC '24). Association for Computing Machinery, New York, NY, USA, 50 – 54.

[35] Joshua Ammermann, Fabian Jakob Brenneisen, Tim Bittner, and Ina Schaefer. 2024. Quantum Solution for Configuration Selection and Prioritization. In Proceedings of the 5th ACM/IEEE International Workshop on Quantum Software Engineering (Q-SE 2024). Association for Computing Machinery, New York, NY, USA, 21 – 28.

[36] Narjes Bessghaier, Mohammed Sayagh, Ali Ouni, and Mohamed Wiem Mkaouer. 2023. What Constitutes the Deployment and Runtime Configuration System? An Empirical Study on OpenStack Projects. ACM Trans. Softw. Eng. Methodol. 33, 1, Article 5 (January 2024), 37 pages.

[37] Qianwen Ye, Wuji Liu, and Chase Q. Wu. 2021. NoStop: A Novel Configuration Optimization Scheme for Spark Streaming. In Proceedings of the 50th International

Conference on Parallel Processing (ICPP '21). Association for Computing Machinery, New York, NY, USA, Article 59, 1 – 10. https://doi.org/10.1145/3472456.3472515.

[38] Yilun Wang, Pengfei Chen, Hui Dou, Yiwen Zhang, Guangba Yu, Zilong He, and Haiyu Huang. 2024. FaaSConf: QoS-aware Hybrid Resources Configuration for Serverless Workflows. In Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24). Association for Computing Machinery, New York, NY, USA, 957 – 969.

[39] Yu Shen, Xinyuyang Ren, Yupeng Lu, Huaijun Jiang, Huanyong Xu, Di Peng, Yang Li, Wentao Zhang, and Bin Cui. 2023. Rover: An Online Spark SQL Tuning Service via Generalized Transfer Learning. In Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23). Association for Computing Machinery, New York, NY, USA, 4800 – 4812.

[40] Yasmine Djebrouni, Isabelly Rocha, Sara Bouchenak, Lydia Chen, Pascal Felber, Vania Marangozova, and Valerio Schiavoni. 2023. Characterizing Distributed Machine Learning Workloads on Apache Spark: (Experimentation and Deployment Paper). In Proceedings of the 24th International Middleware Conference (Middleware '23). Association for Computing Machinery, New York, NY, USA, 151 – 164.

[41] Jingzhi Gong and Tao Chen. 2024. Deep Configuration Performance Learning: A Systematic Survey and Taxonomy. ACM Trans. Softw. Eng. Methodol. 34, 1, Article 25 (January 2025), 62 pages.

[42] Jinhan Xin, Kai Hwang, and Zhibin Yu. 2022. LOCAT: Low-Overhead Online Configuration Auto-Tuning of Spark SQL Applications. In Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22). Association for Computing Machinery, New York, NY, USA, 674 – 684.Jinhan Xin, Kai Hwang, and Zhibin Yu. 2022. LOCAT: Low-Overhead Online Configuration Auto-Tuning of Spark SQL Applications. In Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22). Association for Computing Machinery, New York, NY, USA, 674 – 684.

[43] Joshua Ammermann, Fabian Jakob Brenneisen, Tim Bittner, and Ina Schaefer. 2024. Quantum Solution for Configuration Selection and Prioritization. In Proceedings of the 5th ACM/IEEE International Workshop on Quantum Software Engineering (Q-SE 2024). Association for Computing Machinery, New York, NY, USA, 21 – 28. https://doi.org/10.1145/3643667.3648221

[44] Sibei Chen, Ju Fan, Bin Wu, Nan Tang, Chao Deng, Pengyi Wang, Ye Li, Jian Tan, Feifei Li, Jingren Zhou, and Xiaoyong Du. 2025. Automatic Database Configuration Debugging using Retrieval-Augmented Language Models. Proc. ACM Manag. Data 3, 1, Article 13 (February 2025), 27 pages.

[45] Yi Ding, Ahsan Pervaiz, Michael Carbin, and Henry Hoffmann. 2021. Generalizable and interpretable learning for configuration extrapolation. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021). Association for Computing Machinery, New York, NY, USA, 728 – 740.

[46] Yang Li, Huaijun Jiang, Yu Shen, Yide Fang, Xiaofeng Yang, Danqing Huang, Xinyi Zhang, Wentao Zhang, Ce Zhang, Peng Chen, and Bin Cui. 2023. Towards General and Efficient Online Tuning for Spark. Proc. VLDB Endow. 16, 12 (August 2023), 3570 – 3583.

[47] Zhenyu Yu, Zhibao Wang, Lu Bai, Liangfu Chen, and Jinhua Tao. 2021. Parameter Optimization on Spark for Particulate Matter Estimation. In 2021 Workshop on Algorithm and Big Data (WABD 2021). Association for Computing Machinery, New York, NY, USA, 9–13.