大数据场景中语言虚拟机的应用和挑战

吴明瑜[1,2]，陈海波[1,2]，臧斌宇[1,2]
1. 领域操作系统教育部工程研究中心，上海 200240；
2. 上海交通大学软件学院并行与分布式系统研究所，上海 200240

摘要
语言虚拟机为大数据应用提供了与平台无关的执行环境，简化了应用的开发和部署，因此在大数据场景中得到了较广泛的应用。主要分析了两种主流语言虚拟机——JVM和CLR在大数据场景中的应用，并阐述了使用语言虚拟机面临的4个挑战：初始化及“热身”开销、垃圾回收暂停、异构内存支持、数据格式转换。之后，分别针对4个挑战讨论了现有的解决方案，并分析了这些方案的不足之处及未来可能的优化方向。

关键词
语言虚拟机；垃圾回收；异构内存

中图分类号：TP315　　文献标识码：A　　doi: 10.11959/j.issn.2096-0271.2020035

Applications and challenges of language virtual machines in big data

WU Mingyu[1,2], CHEN Haibo[1,2], ZANG Binyu[1,2]
1. Engineering Research Center for Domain-Specific Operating Systems, Ministry of Education, Shanghai 200240, China
2. Institute of Parallel and Distributed Systems, School of Software, Shanghai Jiao Tong University, Shanghai 200240, China

Abstract
Language virtual machines provide a platform-independent execution environment for big-data applications and simplify their development and deployment phases, so they are widely used in the big-data scenario. The applications of two different kinds of mainstream language virtual machines: JVM and CLR, were analyzed, and four challenges when adopting language virtual machines: initialization and warm-up overhead, garbage collection pauses, heterogeneous memory support, and data layout transformation, were summarized. Afterward, existing approaches to the challenges were discussed and their shortcomings and possible optimizations in the future were analyzed.

Key words
language virtual machines, garbage collection, heterogeneous memory

1 引言

随着大数据时代的到来，对数据分析和处理的需求催生出了一系列大数据处理系统，通过调度集群内的硬件资源，以高并发、分布式的方式处理数据。语言虚拟机（language virtual machine）也被称为管理运行时环境（managed runtime environment），它为应用进程提供了与平台无关的执行环境，使应用具有良好的可移植性，适合在分布异构的环境中使用。同时，语言虚拟机还为应用提供了类型检查、垃圾回收、即时编译等功能，减轻了应用的负担，使开发者能够专注于应用逻辑。由于语言虚拟机的良好特性，很多大数据处理系统使用具有语言虚拟机支持的程序语言编写，如Hadoop[1]、Spark[2]、Flink[3]、DryadLINQ[4]等。其中，为Java提供的Java虚拟机（Java virtual machine，JVM）[5]和为.NET提供的公共语言运行时（common language runtime，CLR）[6]是大数据处理系统的主要选择，其中JVM广泛用于开源的大数据系统，而CLR则主要部署于微软公司的实际生产系统中。

语言虚拟机在为大数据应用开发提供便利的同时，也带来了一系列挑战。语言虚拟机为了构建与平台无关的执行环境，需要经历初始化和“热身”过程，该过程耗时较长，且该过程中的执行效率较低。另外，语言虚拟机引入的垃圾回收（garbage collection）过程一般会要求暂停应用，使得应用执行的时间变长。此外，近年来硬件异构化趋势明显，以非易失性内存（non-volatile memory，NVM）为代表的新型硬件逐渐普及，而语言虚拟机缺乏对这些硬件的支持，因而无法充分利用硬件资源。最后，不同的语言虚拟机的数据存储格式不同，因此虚拟机之间的通信需要额外的格式转换开销。本文将详细阐述语言虚拟机在大数据场景中面临的挑战，并深入探讨已有的解决方案及其存在的不足。

2 大数据场景中语言虚拟机的应用

语言虚拟机的可移植性和安全性等特点为大数据应用的开发和部署提供了便利，因此在大数据场景中得到了广泛使用。本文主要讨论大数据场景中应用较为广泛的两种语言虚拟机：JVM和CLR。这两种虚拟机不仅具有上述优点，还进行了充分的性能优化，因而能为大数据应用提供较为高效的支持。

2.1 JVM

JVM主要是为Java应用提供的语言虚拟机。为了支持与硬件平台无关的执行模式，JVM提供了字节码（byte-code）这一中间表达形式，Java代码首先会被翻译为字节码，然后就可以在任意JVM上执行，从而具备“一次编译，处处执行”的可移植性。JVM提供了基于栈的执行模式，通过构建Java栈依次解释执行字节码，从而完成对Java应用的执行。由于解释执行效率较低，JVM提供了即时编译（just-in-time compilation）功能，会在运行时将频繁使用的代码进行编译优化，从而提高Java应用的整体性能。同时，JVM也提供了自动内存管理（即垃圾回收）以及安全检查功能，进一步提高了易用性和可靠性。

由于开发一个成熟高效的语言虚拟机成本较高（例如，著名的开源JVM实现OpenJDK源码超过1 000万行），很多程序语言没有开发语言虚拟机，而是直接被编译为字节码，在JVM上执行。这些程序语

言被称为JVM语言，其进一步拓宽了JVM的应用范围，丰富了JVM的生态。

目前，在JVM上运行的大数据系统有很多，其中既包含Hadoop、Hive[7]、Hyracks[8]等完全使用Java语言编写的系统，也包含Spark、Flink、Storm[9]等使用JVM语言编写的系统。可以说，JVM在目前的大数据环境中是不可或缺的。

2.2 CLR

CLR是由微软公司开发、为.NET程序提供支持的语言虚拟机。CLR提供了与字节码相似的中间表达——通用中间语言（common intermediate language，CIL）来构建与平台无关的执行环境。同时，CLR也提供了即时编译、垃圾回收、类型安全检查等功能。

CLR主要应用于微软公司内部的大数据系统之中，如Dryad[10]、SCOPE[11]、DryadLINQ等，这些系统已经被部署在微软公司的生产集群中持续提供服务。其中，SQL数据查询是微软公司大数据场景的重点之一，因此微软公司开发了DryadLINQ系统，将面向对象的C#程序与使用LINQ[12]编写的SQL查询语句充分整合，并交由CLR和SQL引擎协同处理。这种混合执行的方式增强了SQL数据查询的表达能力，但也带来了潜在的性能问题。

3 大数据场景中语言虚拟机面临的挑战

图1展示了使用多个语言虚拟机（以JVM为例）执行大数据应用的流程。当接收到任务时，首先需要启动JVM执行环境，而目前JVM的初始化时间较长，还会引入额外的“热身”阶段，这会带来明显的开销；在执行过程中，由于大数据应用对内存的需求量较大，因此会在执行过程中较为频繁地执行垃圾回收过程；此外，尽管现在服务器已经开始采用大容量的非易失性内存设备，但目前JVM缺乏对该硬件的支持，因此无法充分利用内存资源；最后，大数据应用通常需要多个JVM的配合，但JVM之间的通信需要引入复杂的序列化/反序列化过程，以实现数据格式的转换，这增大了通信过程的性能开销。本节将从初始化及“热身”、垃圾回收暂停、异构硬件支持及数据格式转换4个方面探讨语言虚拟机在大数据场景中面临的挑战。

3.1 初始化及“热身”

与编译后即可执行的本地代码（如使用C/C++语言编写的代码）不同，由于语言虚拟机需要为应用构建与平台无关的

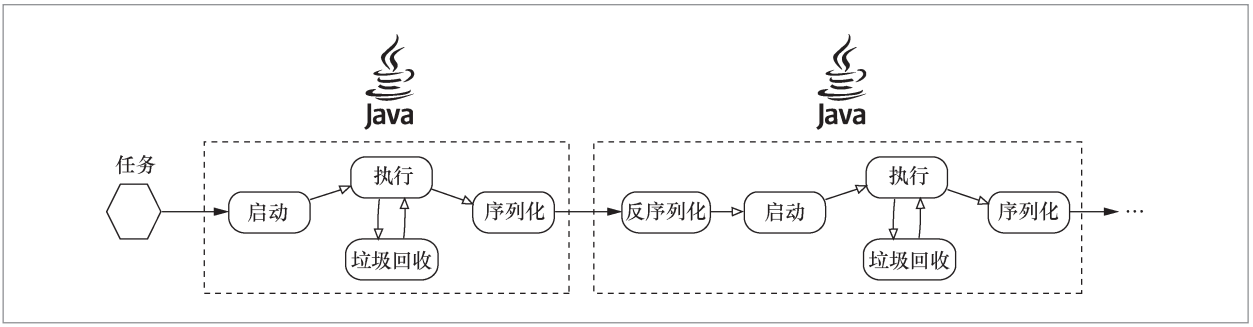

图1 使用语言虚拟机（以JVM为例）的大数据应用执行流程示意图

执行环境，因此在执行应用前要先对虚拟机进行初始化，这往往会带来上百毫秒的开销。在此之后，语言虚拟机还需要经历一个“热身”过程：首先，语言虚拟机往往采用动态加载类型的方式，对于首次使用的类型，虚拟机需要载入其对应的类文件（class file），解析其中的类型信息，并将其转化为类型元数据存储在虚拟机中。由于大数据系统一般比较复杂，使用的类型也较多，因此在运行过程中会出现大量类加载操作，造成性能开销；其次，语言虚拟机的即时编译模块需要对应用的代码进行分析，找出频繁调用的方法和分支，然后对它们进行编译优化，而在优化完成之前，语言虚拟机将采用解释执行的方式运行，其性能远低于编译执行。由于大数据系统的代码量较大，逻辑较为复杂，“热身”阶段一般会持续数秒[13]，因此会显著影响大数据应用的性能。

3.2 垃圾回收暂停

JVM和CLR都提供了垃圾回收模块，用于对应用的内存进行自动管理和回收。由于大数据应用需要处理海量数据，而这些数据都会被转换为对象存储在语言虚拟机的堆内，这会占用较大的内存，并会频繁触发垃圾回收。主流的垃圾回收算法一般会暂停应用以完成内存回收，频繁的垃圾回收会导致应用性能下降。

与其他应用相比，大数据应用特殊的内存行为也进一步影响了垃圾回收的性能。应用的内存行为一般满足“分代假设”[14]，即大多数对象的存活时间较短，只有少部分对象会长时间存活。因此，虚拟机会将自己的堆内存划分为“年轻代”和“年老代”，其中“年轻代”用来保存这些存活时间较短的对象。同时，虚拟机一般会使用“年轻代垃圾回收”来回收“年轻代”的内存，只有在全堆内存严重不足时才会使用“全堆垃圾回收”来回收全堆内存。由于“年轻代”内存只占全堆内存的小部分，因此这种方法能明显降低单次垃圾回收的暂停时间。但是，之前的研究发现[15]，大数据应用的内存行为并不符合这一假设，许多对象能存活较长的时间，导致“年轻代垃圾回收”的效率极低，语言虚拟机不得不通过“全堆垃圾回收”的方式回收内存，因此造成了较长的垃圾回收暂停时间。

3.3 异构硬件支持

为了满足应用的可移植性需求，语言虚拟机已经提供了对部分异构硬件的支持，比如可以在不同的处理器（如x86和ARM处理器）上运行。但是，对于NVM等硬件，其支持还较为薄弱。本文将以NVM为代表进行探讨。

NVM设备是一种具备非易失特性的内存设备，相比普通内存（如动态随机存取存储器（dynamic random access memory，DRAM）），它有很多优点：第一，它具有非易失性，其存储的数据在断电后不会丢失，因此可以用来存储持久化数据；第二，它的容量比普通内存大，单位成本和能耗都比普通内存低。由于大数据系统本身就需要大量内存，NVM的这些特性可以使其以较低的成本扩大语言虚拟机的内存容量，支持更大规模的数据分析。

然而，目前的主流语言虚拟机对NVM的支持非常有限。OpenJDK提出了新的运行时选项，允许应用从NVM创建Java堆[16]，但该选项目前只支持将全堆都创建在NVM上，无法进行细粒度的控制。此外，该选项的提出主要是出于扩容考虑，并没有利用NVM的持久化特性，因此断电后NVM上的数据也将不可用。

3.4 数据格式转换

除了具有虚拟机支持的程序语言以外，一些语言（如Python、R、LINQ等）因为提供了数据分析功能，在大数据场景中也得到了广泛的应用。比如，Spark提供了PySpark[17]和SparkR[18]模块，分别提供对Python和R语言的支持；DryadLINQ则将C#程序和LINQ语句组合在一起，增强了数据分析应用的表达能力。多语言支持丰富了大数据系统的生态，但也带来了数据格式转换的问题。不同的语言会为对象设计不同的数据格式，当数据需要从语言虚拟机管理的程序发送给使用其他语言编写的程序时，就需要进行数据格式转换，反之亦然。由于大数据应用在每个阶段都会处理大量的数据，因此发送过程可能会涉及大量对象的数据格式转换，从而造成明显的开销。

另外，由于每个语言虚拟机对对象的保存格式都不相同，因此语言虚拟机之间的通信也需要进行数据格式转换。在JVM中，这个过程被叫作序列化/反序列化：对象首先被发送者转化为字符串流（序列化），然后经过网络发送给接收者，并由接收者重新转化为对象（反序列化）。研究表明，当数据对象规模较大时，序列化和反序列化的开销将远远超过网络传输的开销，这成为数据通信过程中的瓶颈[19]。

4 现有解决方案

4.1 跳过初始化和“热身”

虚拟机的初始化和“热身”阶段的执行过程具有一定的相似性，之前的工作一般采取跳过部分步骤的方案来降低开销，可大致分为以下4种思路。

- 类型共享。该思路允许多个语言虚拟机共享同类型数据，如果一个虚拟机已经载入了某个类型，那么其他虚拟机就可以直接使用，无须重复加载。比较有代表性的工作是OpenJDK提出的“应用类型数据共享（application class-data sharing，APPCDS）”特性[20]，测试表明，它能将应用的启动时间缩短20%~30%。
- 提前编译（ahead-of-time compilation）。相对于即时编译，提前编译在应用执行前就会编译完成，因此跳过了运行中的编译阶段。IBM公司的JVMOpenJ9已经使用了上述方法，允许应用指定需要提前编译的类型[21]。提前编译可以与类型共享搭配使用，使多个虚拟机共享同一份编译过的代码，从而提升多个虚拟机的启动效率。
- 虚拟机重用。由于JVM的启动时间过长，HotTub系统提出了JVM池的概念[13]：当大数据应用执行完成后，JVM进程不会被销毁，而是会回到JVM池中；如果后续接收到了相似的任务，该JVM会被唤醒，并处理该任务。由于JVM池中的JVM都已经完成了“热身”阶段，因此其执行效率会明显提升。
- 基于系统调用fork。在UNIX操作系统中，fork这一系统调用允许应用从已有进程中创建进程，且新进程的内存状态与原进程完全相同。由于语言虚拟机本身就是系统进程，因此可以通过fork从已有的虚拟机进程中直接创建，而不是从头开始进行初始化[22]。如果能从一个已经充分“热身”的虚拟机中创建其他虚拟机，那么这些虚拟机也能跳过“热身”阶段。

4.2 减少垃圾回收

垃圾回收是大数据场景下的性能瓶颈，

目前有很多工作尝试减少垃圾回收暂停时间，以提高性能。这些工作大致可以分为以下3类。

● 根据内存行为重新设计垃圾回收算法。大数据系统的内存行为具有明显的“阶段性”：例如对于MapReduce一类的大数据应用，其执行过程可以被拆分为一个个“map阶段”和“reduce阶段”。每个阶段都会创建大量临时对象，而在阶段结束时，这些对象将不再存活。由于每个阶段的持续时间可能会达到数秒，因此这些临时对象的存活时间较长，违背了垃圾回收器的“分代假设”。基于这个观察，研究人员提出了基于时代（epoch-based）的垃圾回收器Yak[15]，根据大数据应用中的阶段来划分时代。当一个时代结束时，在该时代中创建的对象被回收。NG2C系统[23]则应用了另一种思路，它将堆划分为N个代$G_0,G_1,G_2,\cdots,G_N$。NG2C系统总是会先回收G_0，只有在回收的内存不够时才会回收G_1，依此类推。同时，NG2C系统会通过提前运行和分析的方式来决定每个对象的分配位置。对于一些存活时间较短的对象，它们会被放入G_0；而对于在大数据系统中某个阶段内创建的对象，它们会被放入之后的代中。这种基于预测的方式减少了盲目的垃圾回收，提高了大数据应用中垃圾回收的效率。

● 脱离语言虚拟机的内存管理。由于传统垃圾回收算法在大数据应用中的性能开销较大，一些工作提出将大数据应用创建的对象搬到堆外，脱离虚拟机的内存管理，从而减少垃圾回收暂停时间。例如FAÇADE系统[24]就将数据处理中需要的对象都保存到JVM的堆外内存中，并设计了新的模块来管理这些内存。Spark的运行时优化系统Tungsten[25]也实现了堆外管理模块，允许大数据应用将存活时间长的对象保存到堆外，从而减轻内存压力。Deca系统[26]则将生命周期相近的对象都转化为字节数组类型，使其脱离垃圾回收的扫描范围，并在这些对象的生命周期结束时统一销毁。

● 应用主动管理内存。一些研究人员认为，大数据应用或系统应当感知自己的内存占用，并采取相应措施减少垃圾回收。ITask[27]系统实现了可中断的大数据任务，当大数据应用监测到垃圾回收较为频繁且回收效率低下时，就会主动暂停一些任务的执行，腾出更多内存空间，以减少垃圾回收带来的开销。类似地，DynMR[28]同样会在内存压力较大时，将部分暂存的数据写回硬盘，以释放内存。

4.3 异构内存支持

由于NVM设备具有持久化、容量大、能耗低等特点，而主流语言虚拟机对NVM的支持比较有限，因此目前已经有研究开始在JVM上为异构内存（DRAM和NVM）提供支持。根据NVM的使用方法，这些研究可以分为以下两类。

● 出于扩容和成本考虑使用NVM。NVM虽然具有以上优点，但与DRAM相比，它存在一定的性能缺陷：其读写时延均高于DRAM，读写带宽均低于DRAM[29]。因此，使用混合内存的系统在性能上比只使用DRAM的系统差。基于以上问题，研究人员提出了基于大数据应用语义的混合内存管理系统Panthera[30]，它以大数据应用中使用的数据集（dataset）为基本单位，通过静态分析获得应用对数据集的访问频率，从而决定其分配位置。例如，部分为了容错而长期缓存的数据集在正常执行时很少被访问，这些数据集就适合放在NVM中。通过使用这一分配策略，Panthera有效利用了NVM的低能耗和大容量特性，同时只造成了较低的性能开销。

● 出于持久化考虑使用NVM。NVM

还可以用来存储持久化数据，以便在断电重启后能够快速恢复。但是，由于语言虚拟机为应用隐藏了硬件平台的信息，应用要完成NVM上数据的持久化比较困难，且会造成较大的性能开销。因此，GCPersist[31]系统提出在垃圾回收过程中由JVM完成数据的持久化，在减轻应用负担的同时降低了持久化开销。另外，GCPersist还通过修改垃圾回收算法，在DRAM和NVM中保存了两份数据，其中NVM中的数据只会在错误恢复过程中使用。这种方法减少了大数据应用在正常运行时访问NVM的次数，从而减少了NVM因读写性能较差带来的开销。

4.4 数据格式转换优化

数据格式转换的优化方案可以分为以下两类。

- 减少数据格式转换的次数。在DryadLINQ程序中，每当程序要在LINQ表达式和C#代码之间切换时，就会发生数据格式转换。为了减少转换开销，Niijima[32]系统采取的方法是对代码进行分析和移动，尽可能地将LINQ表达式和C#代码分别聚合到一起，减少语言切换的次数。Gerenuk[33]系统则对Java代码进行了重编译，使其可以直接操作序列化以后的数据，这样可以减少序列化/反序列化的次数，降低转换开销。
- 降低单次转换的开销。由于Java序列化/反序列化的开销较大，研究人员已经开发出了新的工具来代替Java原生的序列化/反序列化模块[34-35]。这些工具一般会牺牲兼容性（比如不支持Java 6以前的版本），或要求应用手动注册需要序列化的类型，以达到降低序列化开销的目的。Skyway[19]系统则提出了直接传递对象图的技术，通过建立JVM之间的连接，使不同JVM对同一类型对象的解析方式达成共识，之后就可以使用较简单的方式传递对象，简化了序列化/反序列化的过程。

5 现有方案的问题和未来方向

5.1 应用范围受限

目前针对大数据处理的语言虚拟机研究主要集中于批处理类型的场景，应用范围比较有限。比如，Yak、ITask、DynMR针对Hadoop进行分析和优化，而Panthera、GCPersist、Skyway、Gerenuk、Deca等系统则针对Spark进行分析优化。之前的优化方法根据分析结果对应用行为进行假设，但由于分析的应用类型有限，因此其假设可能不具备通用性。比如，Yak的垃圾回收算法假设对象的生命周期与其创建的时代一致，但是在Spark中，用户可以通过cache接口将对象缓存在JVM中，这些对象的生命周期要长于其创建的时代，因而违背了Yak的假设。对于流处理、图查询、近似计算等大数据场景，它们的应用行为与批处理存在较大不同，如单个任务的执行时间更短、数据分析和存储的粒度更小等，而这些应用类型目前在语言虚拟机上的行为还缺乏深入的分析。因此，未来的工作需要进一步拓宽应用范围，为更多类型的应用提供高效的语言虚拟机支持。

5.2 可移植性和安全性降低

Deca、FAÇADE、Gerenuk等系统对大数据应用中的数据和代码进行了转化，使其更多地依赖于本地数据格式和本地代码执行，以提高性能。但是，由于这种方法生成的代码和数据都脱离了JVM的管

理，因此不再具有可移植性，还会带来潜在的安全性问题。例如，FAÇADE将数据放到堆外进行管理，并设计了专门的数据结构来解决类型继承、线程间同步、内存回收等问题，这部分代码直接使用类似指针的方式操作数据，不再具有类型安全特性，因此造成了潜在的安全风险。针对这一问题，OpenJDK社区发起了瓦尔哈拉项目（project Valhalla）[36]，其目标之一就是为Java提供“值类型（value object）”。“值类型”是由用户自定义的类型，其数据存储格式与本地格式相似，但可以使用Java代码进行访问和管理，因此仍具有可移植性和类型安全保证。“值类型”可能会在未来进入OpenJDK主线，为大数据场景提供新的支持。

5.3 数据格式转换难以根除

由于不同语言各有所长，因此在大数据场景中使用多种语言进行数据处理已经成为比较普遍的现象。由于不同语言对数据格式的定义不同，一旦进行跨语言通信，数据格式转换就难以避免。目前的解决方案只能减少数据格式转换的次数或者降低单次转换的开销，并不能完全消除耗时的数据格式转换阶段。一种可能的消除格式转换的方法是对现有的语言进行整合，使它们能够统一运行在相同的语言虚拟机之上，从而可以共享相同的数据格式，免除数据转换开销。OpenJDK社区提出的GraalVM[37]项目就提供了这样一种语言虚拟机，它能支持包括Scala、R、JavaScript、C、C++在内的多种语言，为消除数据格式转换提供了可能，但目前还没有针对GraalVM上数据格式转换的研究。而对于不同虚拟机之间数据表示方式不同的问题，可以对OpenJDK已有的APPCDS特性进行扩展，使多个虚拟机就数据格式达成一致，进而避免虚拟机间的数据格式转换。

6 结束语

语言虚拟机具有良好的可移植性和安全性，还提供了垃圾回收和即时编译等功能，能简化大数据应用的开发和部署，因此在大数据场景下得到了较为广泛的应用。本文主要介绍了两种主流的语言虚拟机（JVM和CLR）在大数据场景中的具体应用，并从初始化和“热身”、垃圾回收、异构硬件支持、数据格式转换4个方面分析了语言虚拟机在大数据场景下面临的挑战和现有的解决方案。本文同时指出，已有的解决方案还存在应用领域受限、可移植性和安全性降低、格式转换难以根除等问题。在大数据处理方式多样化的趋势下，语言虚拟机还存在巨大的分析和优化空间。

参考文献:

[1] WHITE T. Hadoop: the definitive guide[M]. Sebastopol: O’Reilly Media, Inc., 2012: 647.

[2] ZAHARIA M, XIN R, WENDELL P, et al. Apache Spark: a unified engine for big data processing[J]. Communications of the ACM, 2016, 59(11): 56-65.

[3] CARBONE P, KATSIFODIMOS A, EWEN S, et al. Apache Flink: stream and batch processing in a single engine[J]. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2015, 36(4): 28-38.

[4] YU Y, ISARD M, FETTERLY D, et al. DryadLINQ: a system for general-purpose distributed data-parallel computing using a high-level language[C]// The 8th USENIX Symposium on Operating Systems Design

and Implementation. Berkeley: USENIX Association, 2008: 383–400.

[5] LINDHOLM T, YELLIN F, BRACHA G, et al. The Java virtual machine specification[M]. London: Pearson Education, 2014: 584.

[6] KENNEDY A, SYME D. Design and implementation of generics for the .NET common language runtime[C]// The ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation. New York: ACM Press, 2001: 1–12.

[7] DU D Y. Apache Hive essentials[M]. Birmingham: Packt Publishing Ltd, 2015: 191.

[8] BORKAR V, CAREY M, GROVER R, et al. Hyracks: a flexible and extensible foundation for data-intensive computing[C]// The IEEE 27th International Conference on Data Engineering. Piscataway: IEEE Press, 2011: 1151–1162.

[9] IQBAL M H, SOOMRO T R. Big data analysis: Apache Storm perspective[J]. International Journal of Computer Trends and Technology, 2015, 19(1): 9–14.

[10] ISARD M, BUDIU M, YU Y, et al. Dryad: distributed data-parallel programs from sequential building blocks[C]// The 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems. New York: ACM Press, 2014: 59–72.

[11] CHAIKEN R, JENKINS B, LARSON P, et al. SCOPE: easy and efficient parallel processing of massive data sets[J]. Proceedings of the VLDB Endowment, 2008, 1(2): 1265–1276.

[12] PIALORSI P, RUSSO M. Introducing microsoft® LINQ[M]. Redmond: Microsoft Press, 2007: 282.

[13] LION D, CHIU A, SUN H, et al. Don't get caught in the cold, warm-up your JVM: understand and eliminate JVM warm-up overhead in data-parallel systems[C]// The 12th USENIX Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2016: 383–400.

[14] UNGAR D. Generation scavenging: a non-disruptive high performance storage reclamation algorithm[C]// The 1st ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments. New York: ACM Press, 1984: 157–167.

[15] NGUYEN K, FANG L, XU G Q, et al. Yak: a high-performance big-data-friendly garbage collector[C]// The 12th USENIX Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2016: 349–365.

[16] OpenJDK. JEP 316: heap allocation on alternative memory devices[Z]. 2016.

[17] DRABAS T, LEE D. Learning PySpark[M]. Birmingham: Packt Publishing Ltd, 2017: 250.

[18] VENKATARAMAN S, YANG Z H, LIU D, et al. SparkR: scaling R programs with Spark[C]// The 2016 International Conference on Management of Data. New York: ACM Press, 2016: 1099–1104.

[19] NGUYEN K, FANG L, NAVASCA C, et al. Skyway: connecting managed heaps in distributed big data systems[C]// The 23rd International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2018: 56–69.

[20] OpenJDK. JEP 310: application class-data sharing[Z]. 2017.

[21] PIRVU M. Optimize JVM start-up with Eclipse OpenJ9[R]. 2018.

[22] WANG K A, HO R, WU P. Replayable execution optimized for page sharing for a managed runtime environment[C]// The 14th EuroSys Conference 2019. New York: ACM Press, 2019: 1–16.

[23] BRUNO R, OLIVEIRA L P, FERREIRA P. NG2C: pretenuring garbage collection with dynamic generations for HotSpot big data applications[C]// The 2017 ACM SIGPLAN International Symposium on Memory Management. New York: ACM Press, 2017: 2–13.

[24] NGUYEN K, WANG K, BU Y Y, et al. FAÇADE: a compiler and runtime for (almost) object-bounded big data applications[C]// The 20th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2015: 675-690.
[25] XIN R, ROSEN J. Project Tungsten: bringing Apache Spark closer to BareMetal[R]. 2015.
[26] SHI X H, KE Z X, ZHOU Y L, et al. Deca: a garbage collection optimizer for in-memory data processing[J]. ACM Transactions on Computer Systems, 2019, 36(1): 1-47.
[27] FANG L, NGUYEN K, XU G Q, et al. Interruptible tasks: treating memory pressure as interrupts for highly scalable data-parallel programs[C]// The 25th Symposium on Operating Systems Principles. New York: ACM Press, 2015: 394-409.
[28] TAN J, CHIN A, HU Z Z, et al. DynMR: dynamic MapReduce with reducetask interleaving and maptask backfilling[C]// The 9th European Conference on Computer Systems. New York: ACM Press, 2014: 1-14.
[29] YANG J, KIM J, HOSEINZADEH M, et al. An empirical guide to the behavior and use of scalable persistent memory[C]// The 18th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2020: 169-182.
[30] WANG C X, CUI H M, CAO T, et al. Panthera: holistic memory management for big data processing over hybrid memories[C]// The 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM Press, 2019: 347-362.
[31] WU M Y, CHEN H B, ZHU H, et al. GCPersist: an efficient GC-assisted lazy persistency framework for resilient Java applications on NVM[C]// The 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. New York: ACM Press, 2020: 1-14.
[32] XU G Q, VEANES M, BARNETT M, et al. Niijima: sound and automated computation consolidation for efficient multilingual data-parallel pipelines[C]// The 27th ACM Symposium on Operating Systems Principles. New York: ACM Press, 2019: 306-321.
[33] NAVASCA C, CAI C, NGUYEN K, et al. Gerenuk: thin computation over big native data using speculative program transformation[C]// The 27th ACM Symposium on Operating Systems Principles. New York: ACM Press, 2019: 538-553.
[34] KRYO. Java binary serialization and cloning: fast, efficient, automatic[R]. 2020.
[35] COLFER. The Colfer serializer[R]. 2017.
[36] OpenJDK. Valhalla[R]. 2020.
[37] GraalVM. Run programs faster anywhere[R]. 2020.

作者简介

吴明瑜（1993- ），男，上海交通大学软件学院博士生，主要研究方向为语言虚拟机和非易失性内存。

陈海波（1982- ），男，博士，上海交通大学教授、并行与分布式系统研究所所长，领域操作系统教育部工程研究中心主任，国家杰出青年基金获得者，国际计算机学会（ACM）杰出科学家，中国计算机学会（CCF）杰出会员与杰出演讲者，主要研究方向为操作系统和系统安全。曾获教育部技术发明奖一等奖（第一完成人），全国优秀博士学位论文奖、CCF青年科学家奖。目前担任ACM SIGOPS ChinaSys主席、CCF系统软件专业委员会副主任、*Communications of the ACM*中国首位编委与Special Sections领域共同主席、*ACM Transactions on Storage*编委、《大数据》期刊编委。曾任ACM SOSP 2017大会共同主席、ACM CCS 2018系统安全领域主席、ACM SIGSAC奖励委员会委员。研究工作获得华为技术有限公司最高个人贡献奖、Google Faculty Research Award、IBM X10 Innovation Award、NetApp Faculty Fellowship等企业奖励。

臧斌宇（1962- ），男，博士，上海交通大学教授、软件学院院长。2011年全国优秀博士学位论文指导教师，2015年“挑战杯”全国竞赛特等奖指导教师。兼任国务院学位委员会软件工程学科评议组成员、教育部高等学校软件工程专业教学指导委员会副秘书长、全国工程教育专业认证专家委员会计算机类专业认证分委员会委员、CCF杰出会员、国家示范性软件学院联盟副理事长。主要从事系统软件方向的研究，致力于计算机核心课程的教学改革。近年来在SOSP、USENIX ATC、Eurosys、ASPLOS、ISCA、HPCA、PPoPP等国际会议上发表论文20余篇。主持多项国家级科研项目。

收稿日期： 2020-06-17
基金项目： 国家自然科学基金资助项目（No.61672345）
Foundation Item: The National Natural Science Foundation of China (No.61672345)