華中科技大學

本科生毕业设计（论文）参考文献译文本

Moshe Babaioff, Yishay Mansour, e.t.c. ERA: A Framework for Economic Resource Allocation for the Cloud. WWW '17 Companion Proceedings of the 26th International Conference on World Wide Web Companion, Pages 635-642.

院　系 电子信息与通信学院

专业班级 电子信息工程

（基于项目信息类专业教育实验班（2+2））201401

班

姓　名 黄 振

学　号 U201414180

指导教师 钟国辉

2018 年 3 月

译文要求

一. 译文内容须与课题（或专业内容）联系，并需在封面注明详细出处。

二. 出处格式为

图书：作者.书名.版本（第×版）.译者.出版地：出版者，出版年.起页～止页

期刊：作者.文章名称.期刊名称，年号，卷号（期号）：起页～止页

三. 译文不少于5000汉字（或2万印刷符）。

四. 翻译内容用五号宋体字编辑，采用A4号纸双面打印，封面与封底采用浅蓝色封面纸（卡纸）打印。要求内容明确，语句通顺。

五. 译文及其相应参考文献一起装订，顺序依次为封面、译文、文献。

六. 翻译应在第七学期完成。

译文评阅

导师评语

应根据学校“译文要求”，对学生译文翻译的准确性、翻译数量以及译文的文字表述情况等做具体的评价后，再评分。

评分：_____（百分制）　　指导教师（签名）：_____

年　月　日

题目
ERA－适用于云计算的经济资源分配框架

摘要

从系统的角度来看，云计算已经非常成熟了，但目前所部署的解决方案所采用的经济机制相当初级，因此导致硬件资源的分配并未达到最佳标准，而这些资源是非常昂贵的。

本文中，我们提出了ERA，一个用于调度和计价云资源的完整框架，旨在基于经济准则来分配资源，从而提高云资源使用的效率。ERA架构细致地抽象了底层的云基础设施，使得调度和定价算法的开发，可以独立于具体的底层云基础设施，和这些基础设施所关注的细节。具体来说，我们设计的ERA，可以灵活地配置于任何云系统的上层，也可以配置于这些接口的上层——给云资源管理者的接口，以及给用户（这些用户预定资源来运行他们的作业）的接口。用户提交的作业是根据预期需求动态计算的价格来排定调度的。此外，ERA为可插拔的算法模块提供了一个关键的内部应用程序接口，这些模块涵盖调度、定价和需求预测。我们提供了原型软件，通过将其部署在公有云和私有云上进行测试（包括微软的Azure Batch和Hadoop/YARN），证明了该架构的有效性。

一个更长远的想法是，我们希望促进经济学和系统学之间的交叉学科协作。为此，我们开发了一个仿真平台，通过这个平台，经济学专家和系统专家可以测试他们的算法实现。

关键词

云计算；经济；动态定价；预定

1. 介绍

云计算，不管是私有云还是公有云，在业界都是实现高投资回报比的范例，这是通过共享庞大的计算基础设施实现的，而建造和操作这些基础设施的成本很高[5,14]。高效的共享中枢围绕着两个关键要素：1）能够安全高效地复用一系列硬件资源，为不同用户提供服务的一套系统基础设施；以及2）在多个用户的资源需求发生冲突时可以进行仲裁的经济机制。

最先进的云产品提供上述两方面的解决方案，但不同程度上都具有一定的复杂性。系统方面的挑战一直受到广泛研究，重点是时空复用。空间复用包括在租户之间共享服务器，其安全性由虚拟机[34,32,7]和容器技术[23,22]保障。时间复用包括一系列根据时间排定任务的技术和系统。从严格支持服务水平目标（SLO）[10,17,30,11]，到最大化集群利用[12,35,18,25,24,13,8]，都是研究的重点。其中许多进展已经在公有云[26,4]和私有云[1,31,15,33,8]上部署了解决方案。这表明在云端如何解决系统挑战已相对成熟。另一方面，尽管在最近的研究中经济挑战收到

了一些关注（见[27,16,6,28,19,20]和其中的引用）,但这些基本经济准则尚未转化为实践中可行的解决方案。

1.1 经济挑战和ERA的方法

在现有的云环境中，资源分配由相当初级的经济机制运作。第一种（常见于私有云[31,8,15,33]）是预先支付的固定保障配额机制。第二种机制（常见于公有云[26,4]）是基于单位价格按需分配：用户为其使用的每一单位资源支付实实在在的金钱（或者，在公司内部支付法定货币）。在大多数情况下，这些都是固定价格，而值得注意的一个例外是亚马逊的动态定价的现货实例。（尽管根据独立分析来看，这些现货实例也并非真正利用市场机制来确定价格[3]。）但是，现货实例产品没有提供保障服务，以至于如果用户出价太低，这些实例可能被销毁。因此，用户在购买使用现货实例时，需要特别关注竞价，还需要考虑到可能不太适用于一些高优先级的生产作业[21,28,2]。最根本的问题是，我们需要找到一个能够高效实现预期的高产出的定价和调度方案。

效率：从经济学的角度来看，云系统最基本的目标是最大化经济效益，即最大化所有的用户从系统中获得的价值总和。例如，当两个用户有需求冲突时，其中改变任务运行时间、地点甚至取消任务所付出的代价更低的那位用户，应当在此次冲突中让步——切换至备选方案。资源将被分配给具有“最高边际价值”的用户。对于一个给定的云，最理想的分配方案的基准是，有一个“无所不知”的调度器：有权访问所有用户的所有信息，这些信息包括他们的内部成本和替代方案；有权以一种能够最大化云所有者的效益的方式，决定将什么资源分配给什么人。再次强调：要获得有意义的效益衡量，我们必须计算获得的价值而非使用的资源，而且我们应该将最大化这一基于价值理念的效益作为目标。（当然，另一个重要目标是收益，但是我们注意到潜在收益受限于创造的价值，因此取得高效益是获得高收益所需要的。此外，由于云服务提供商之间存在竞争，这些提供商通常着眼于提高短期效益，因为这可能会产生正面的长期收益效应。在提供高水平服务的“平台即服务”（PaaS）中，增加收益的讨论通常被指责。本质上这与分配效益是正交的，超出了本文的范畴。）

当前解决方案的局限性：秉持着基于价值的效益理念，我们评估了常见的被部署应用的计价机制。私有云框架[31,8,15,33]，典型地依靠预先支付的保障性配额。这种方案的主要问题是：形式上，没有提供真正的公用资源共享——为了保证每位用户总是能获得可用的预付费的保障性资源，云系统实际上必须保持充足的资源，以满足其所允诺的负载总和，尽管在任一时刻可能只有一小部分资源被使用。类似这种“工作保留 公平排队”[12]的机制，通

常为提高利用率而设计，但是他们没有在根本上改变基于价值的效益方程，因为超出用户配额提供的资源通常没有分布式开销，也没有保障。此外，一次性预付费说明用户使用他们的受保障的资源的边际成本本质上是零，因此不管他们有没有做“有用”的作业来充分利用资源，他们都倾向于使用这部分资源来做一些“无用”的作业。这经常导致云系统看似满载运行，从工程的每个角度看都是如此，但从经济的角度来看，因为大多数时间的大多数作业价值非常低，所以整个系统实际上是运转在一个非常“低生产力”的状态中的。

另一方面，公有云产品[26,4]通常使用按需单位定价机制。这一解放方案的问题在于，资源的可用性无法提前保障。通常，需求是“尖峰”的：短时高频需求散布于长时低频需求之中。这样尖峰的需求在其他共享基础设施的领域也很典型，例如计算机网络带宽、电力系统、滑雪度假区等。所有这些案例中，共享资源的提供商都面临这样一个两难情境：是提供极其昂贵的容载冗余供应，还是放弃高峰期的保障性服务？在这些云系统中，对于足够重要的作业，用户无法承担作业不能运行的风险，导致按需定价只适合低价值或者时间高度灵活的作业，而大多时间不灵活的重要生产作业仰仗于购买长期的保障性资源权限。尽管灵活的单位价格（例如亚马逊的现货实例）可能比固定单价有优势，因为前者可以在时间上平滑需求，但如前文所强调，前者仅仅有机会服务于典型的低价值非生产作业。

ERA的方法：我们在ERA中提供的定价模型可以实现资源共享和需求平滑，对于高价值的生产作业也不例外。这是通过预定来实现的，这一理念众所周知，常用于酒店的房间或超级计算机的时间、还有一些云系统[10,17,30]等众多类型的资源。但我们的预定，在定价和调度方面都以一种灵活的方式实现。我们专注于调度和定价批处理风格的计算所面临的经济挑战，这些计算在共享的云基础设施上是能完全满足时间水平服务目标（即截止日期这一关键指标）的。呈现给用户的基本模型是资源预定。在“预定的时间”，用户程序指定预定请求。这种请求的基本形式是：（请求的一般形式是由ERA的“竞价描述语言”（见2.3.2）给出的，竞价描述语言允许指定多项资源、有关资源使用的随时间变化的变量“shapes”，及其组合）

基本预定：我需要100个容器（每个容器有6GB的内存和2个CPU核心），5个小时，今天6:00到18:00之间的某个时间段，我最多愿意付100美元。

这类工作负荷非常突出，许多“大数据”就属于这个类别[11,30,17]。而且这为我们提供了一个进行经济调查研究的独特机会。尽管最先进的解决方案为资源共享提供了高效的系统级机制，这些方案仍然依赖于用户能够出于好心，向系统完全真实地说明他们的资源需求和截止日期。ERA的机制确保用户支付的最终价格是系统为授予请求所能提供的最低价格。用户的灵活性越高，用户获得理想价格的机会就越大。如果这个最低价格超过了用户

愿意支付的最高价格，那么请求将会被回绝。（或者，ERA的机制可以将这个最低价格作为“报价”呈现给用户，用户可以决定是接受还是拒绝）一旦预定请求被接受，在预定时间内所需付款金额是固定的，用户可以确信在所请求的时间窗口内，他所预定的资源是可用的。这一保证是满足请求，而不是对特定时间的特定资源提供承诺。有关向用户提供的模型的更多细节，参见第2.3.1节。

1.2 ERA总览

制定一个优秀的云资源分配方案这一挑战中的关键部分是云资源的多面性：虽然我们的目标是在比较高层次的经济层面，但是必须直接在计算机系统工程层面进行具体实现。我们必须使用巧妙的算法连接这极为不同的两个视角，并使用合适的软件工程来实现。事实上，在关于云系统的文献中，可以看到许多论文分别解决这些方面的其中一方面，包括“系统”的论文以及关于在云系统中调度和定价作业的理论的论文。遗憾的是，文献中这些不同的思路常常不能彼此联结，也很难组合成一个整体的方案。我们认为在这一点上的一个关键挑战是提供一个包含所有这些考虑因素的共同抽象。我们称之为架构挑战。

我们应对这一挑战的答案就是ERA系统（经济资源分配）。我们将ERA系统设计为云用户和底层云基础设施之间的中间层。ERA提供了一个单一模型，涵盖了云系统底层所有内容各异的关键问题：经济的、算法的、系统级的和人机接口的等等。ERA旨在通过经济原则的方式指导云系统的资源分配决策，从而将经济学见解实际上集成到现实世界的系统基础设施中。（并非云上的所有资源都必须由ERA管理。云也可能让ERA只管理所有资源的一个子集（允许系统进行增量测试），或者将有几个ERA的实例来管理资源的不同子集。）这是通过关键架构抽象实现的：ERA的云接口，隐藏了资源管理基础设施的许多细节，使ERA几乎能够独立于底层云基础设施来应对经济挑战。ERA满足三个关键设计目标：1）提供了一个清晰的理论抽象，能够进行更正式的研究；2）是一个实用的端到端软件系统；3）为扩展性而设计，所有的算法都易于演进和实验。

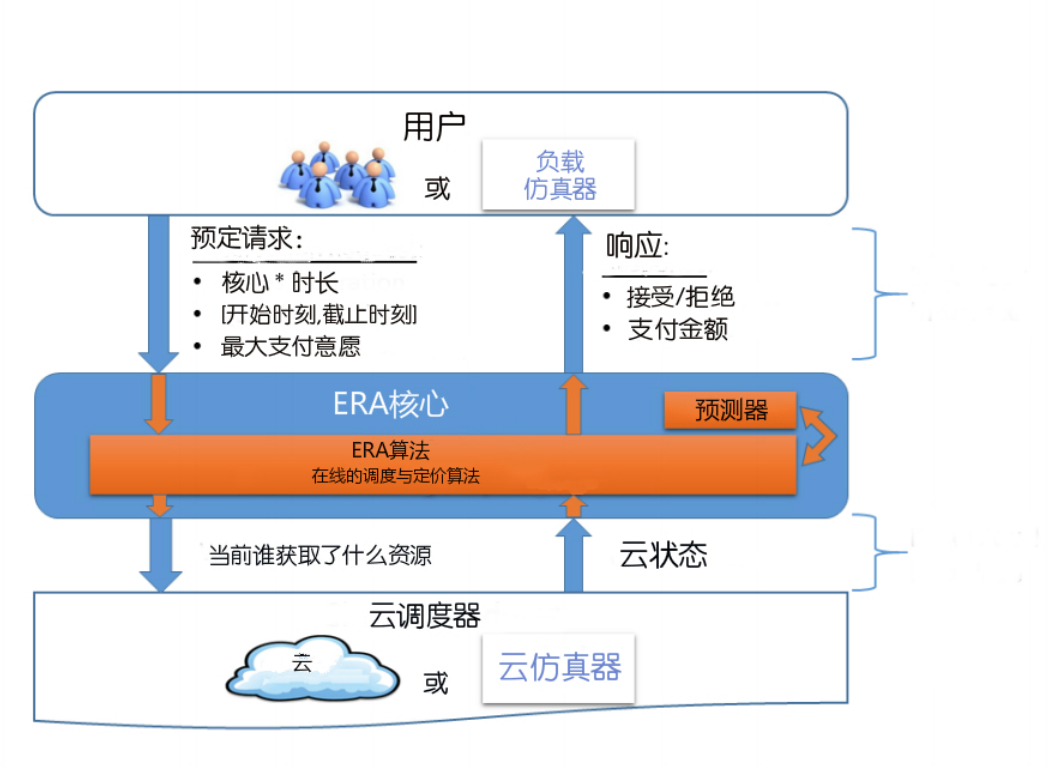

图1 ERA架构。ERA系统被设计为用户和底层云基础设施之间的中间层。同样的实际核心代码也可以与仿真器组件对接交互。

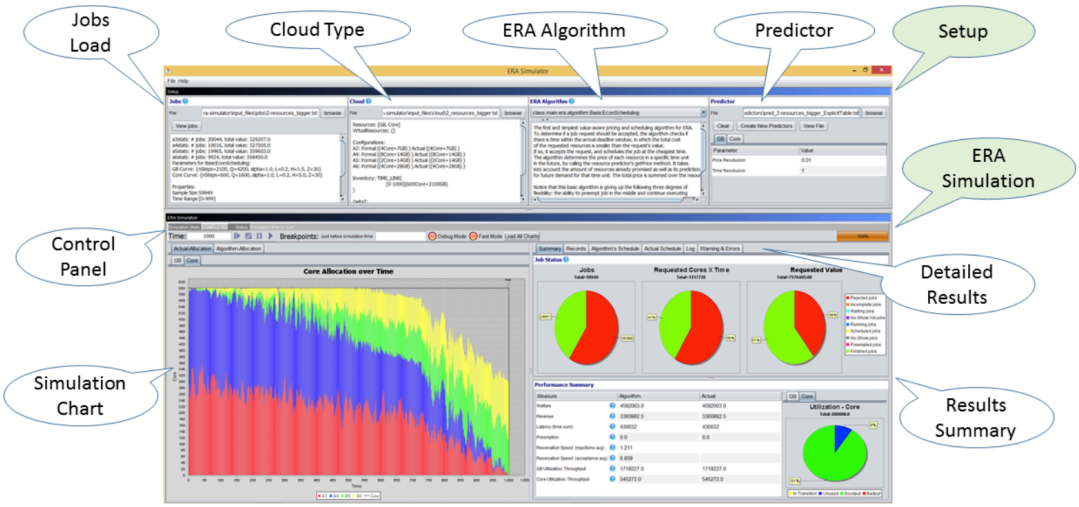

图2 ERA仿真器屏幕截图

ERA的关键应用程序接口：ERA有两个主要面向外部的应用程序接口，以及一个关键的内部应用程序接口。图1给出了ERA架构的高级示意图。第一个外部应用程序接口面向用户，并为他们提供上述云服务的经济预定模型。第二个外部应用程序接口面向低级云资源管理者，对其所关注的问题进行解耦，将基础云系统从任何时间相关的调度或定价问题中解放出来，并且让ERA系统不用承担：以合理的资源区域分配方式，将指定的处理器分配

给指定的任务；低级别的进程启动、进程交换机制。更多细节见第2.3节。

最后，内部应用程序接口是可插拔的调度、定价和预测算法模块。我们的基本调度和定价算法根据供需情况动态计算未来的资源价格，其中需求既包括已经提交确认的资源，也包括所预测的未来请求的资源，并尽可能以最低价对当前需求进行计划和定价。我们的基本预测模型使用历史的运行记录来估计未来的需求。于是，这一灵活的算法应用程序接口易于进行算法、学习和经济方面的优化。第3节介绍了内部接口以及我们的基本算法实现。

我们定义这种抽象的目标，比ERA系统中单纯的软件工程更为深远。为了促进融合系统和经济两方面的考虑，我们还构建了灵活的云仿真框架。仿真器提供关键指标的评估，包括负载或延迟等“系统指标”以及福利或收益等“经济指标”，并提供结果可视化（见图2中的截屏）。该仿真器旨在为云系统管理人员和研究人员提供便利的工具。云系统管理人员可能想要评估ERA的性能，以作为实现系统整合应用的一个步骤；研究人员为ERA开发新算法，想要在不运行大型集群的情况下测试他们的实现。如图1所示，接收真实用户请求并通过底层云资源管理器运行的核心代码，可以连接到仿真器而不是真实用户，这就方便在不同负载和备选云模型下对其进行测试。通过比较我们的仿真器和物理集群运行的结果，我们发现仿真器是可信的（第4节）。

ERA系统是用Java实现的，而用C#写的一个替代实现（ERA的一个子集）也已经完成。我们在仿真器中对ERA做了大量的运行测试，以及在公有云和私有云中使用两个优秀的资源管理器进行了原型软件运行测试：整个系统与Hadoop/YARN[31]进行接口交互，C#版本的代码与微软的Azure Batch仿真器（Azure Batch是一种云规模的作业调度和计算管理服务。 https://azure.microsoft.com/en-us/services/batch/）进行交互，并在上面测试。这些运行测试表明ERA算法能够成功提高云的使用效率，并且ERA是可以成功整合到真实的云系统中的。另外，我们证明了ERA仿真器可以很好地近似真实系统上的实际运行状况，因此可以作为开发和测试新算法的有用工具。在第4节中，我们将介绍这些运行的其中一些结果。

贡献：总而言之，我们提出了ERA，一个用于定价和调度，且满足全时段服务水平目标的预定系统。ERA做出了如下贡献：

1. 我们提出了一个抽象和系统架构，使我们能处理与底层云基础设施相关的经济挑战。 2. 我们设计了算法用于调度和定价达到服务水平目标的批处理作业，以及预测资源需求。

3. 我们设计了一个可靠的云仿真器，系统专家和经济专家可以通过它研究和测试算法实现。

4. 我们将ERA和两个云基础设施集成在一起，并通过实验展示了其有效性。

2. ERA的模型和架构

2.1 使用动态价格的竞价预定模型

ERA旨在处理云系统的一组计算资源，如核心和内存，其目标是高效地将这些资源分配给用户。其基本思想是，用户可以对所需资源进行预定，以在将来某个时间点运行作业。并且，一旦接受预订，就可以保证这些资源（尽可能地）是在所预定的时间内可用的。对所预定资源的可用性保证，使得高价值的作业可以像预付费保证配额模型那样使用云，而无需购买整个容量（对所有时段），因此，ERA也允许资源的时间共享，这增加了效益。

这些资源的价格作为预定的时间内的报价，并根据（算法估计的）需求和供应变化动态计算。用户在时间方面越灵活，则可以自动获得越低的价格。基本思想就是这些动态价格将调节需求，更好地利用云资源。这种机制还可以确保在高峰时期——需求根本无法满足——运行的作业是最有价值的，而不是任意的。ERA使用一种简单的竞价模型，用户在每个作业请求中附加一个货币值，指定用户愿意为运行该作业而支付的最大金额。不能适应高峰期的作业所损失的价值，可作为购买额外云资源所能获得的价值的量化，这将是云服务提供商采购决策流程的重要参照指标。

2.2 云模型

ERA框架中的“云”，建模为一个售卖多种资源的实体，该实体绑定一些配置，资源的容量可能随时间而改变。这些配置项，以及“虚拟资源”的新概念，是用来表示云的“约束条件”，例如打包约束。特别地，“云”被定义为：（1）一组待售的正式资源（例如核心或GB）。我们还允许通过使用“虚拟资源”的概念来捕获底层基础设施的其他限制；（2）一组正式资源配置：每个配置由一组资源定义（例如，“配置A”等价于“4核心和7GB”），（这些配置是预设的，但是ERA的云模型也支持每个作业通过定义资源的配置（例如，“配置B”等价于“单个核心”）选择所需要的一组资源（如YARN）。ERA的云模型也反映了系统为满足这些配置所需要的实际资源。实际资源通常比正式资源要更大。我们应该利用这个差距，对复杂系统分配资源时所面临的过载开销进行建模。实际资源可以由正式资源和虚拟资源构成；（3）库存：不同时刻的资源量（包括正式的和虚拟的）；（4）云的时间定义（例如，云的时间单位精确度等）。

2.3 ERA的架构

ERA系统是用户与云调度器之间的一个智能层，如图1所示。系统接收到用户发出的作业预定请求的流。每个请求都会描述用户希望预定的资源及其时间范围，以及用户愿为这些资源支付的最高价格。ERA通过这些请求中的一部分，以实现总价值（收益）最大化。第2.3.1节描述了这些用户请求的接口。

ERA与云调度器交互，以兑现所允诺的预定资源。ERA指挥云如何为作业分配资源，云应该能够遵循ERA的指挥，以及提供内部状态（例如可用资源的容量）的更新（这是可选的），从而允许ERA重新规划并优化。第2.3.2节描述了ERA与云调度器的接口。

ERA架构在算法模块中封装了调度和定价的逻辑。算法使用预测模块根据预期需求和供应状况动态计算价格。该架构能够切换不同算法，并应用不同的学习方法。第3节介绍了ERA与算法组件的内部接口。

2.3.1 ERA与用户之间的接口

ERA与用户之间的接口处理用户的预定请求流，确定以什么价格接受什么预定。

竞价描述语言

每个预定请求根据ERA的竞价描述语言对资源进行竞价投标。竞价描述语言是[10]中正式定义的预定定义语言的一个扩展。竞价由一列资源请求和这列请求的最大意愿支付价格组成。每个资源请求指定了所请求的资源的配置，需要这些资源的时长，以及一个时间窗口[开始时刻，截止时刻)。所有资源必须在开始时刻之后（包括该时刻），截止时刻之前（不包括该时刻）完成分配。例如，一个资源请求需要3单位的配置A，2单位的配置B，一共2小时，今天早上6点到下午6点之间。这里说的配置A，配置B，见第2.2节所述。

通过支持一列资源请求，ERA可以描述更复杂的作业，包括将每个请求分解为资源单元，以运行MapReduce这样的分布式作业，亦或者子啊某种程度上指定请求的顺序。只有当列中的所有资源请求都通过时，当前的ERA算法才能接受该作业——也就是对多个资源请求应用“与”运算符。更复杂的版本可能支持更复杂更丰富的竞价描述，例如支持其他运营商和递归竞价。为表达清晰，在本文中我们介绍简单版本的ERA，预定请求是单个资源请求，资源配置只有一个。

*makeReservation*方法

与用户预定请求之间的接口由单个“makeReservation”方法组成，该方法处理由某个用户发送的作业预定请求。ERA可以接受请求并定价，也可以拒绝请求。该方法的基本输入参数是作业的竞价和标识符。该竞价封装了资源请求列表，以及用户愿意为此支付的最高价格（如上所述）。该方法的输出是对请求的接受或拒绝，以及需要向用户收取的费用（如

果接受）。（或者，系统可以允许在作业运行完成后付款（取决于那时候的系统负载），也可以更加灵活些：既考虑预定的资源量，又考虑实际使用的资源量。）

接受作业去请求的主要作用就是确保用户在预期的时间范围内的某个时间可以获得所需的资源。已被接受的作业，在请求的窗口期开始时刻必须就绪，以使用所有请求的资源。只要ERA在窗口期内提供了所请求的资源，我们就可以认为该请求已满足。

2.3.2 ERA与云的接口

ERA和云调度器的接口由两个主要方法构成。这个接口使得云可以获得有关应在当前申请的资源的分配信息，并向ERA提供作业实际执行情况和云资源变更的反馈信息。

*getCurrentAllocation*方法

这个方法是ERA和实际的云调度器之间的主要接口。云应该反复调用该方法（通常也就是说，每隔几秒），并要求ERA提供当前的分配信息。（出于性能考虑，也可以用ERA的事件驱动方案代替该轮询方案：发生新的分配时，ERA将该分配事件发送到云调度器）。该方法返回一个“分配”，包含应当立即分配资源的作业列表，及其所需配给的资源情况。在单一资源的简单情况下，这个“分配”是“作业J现在应获取W资源”的一个列表。实际的云基础设施应更新当前分配给所有作业的资源，以满足新的分配。这个新分配直到下一个查询返回更新的分配之前都是有效的。底层云调度系统需要足够频繁地轮询ERA，并尽快执行新的分配，以便任何更改都能在合理的低延迟内生效。ERA系统的主要职责是确保该查询的返回结果序列能适用于所有已接受的预定。

*update*方法（可选使用）

云可以使用该方法定期更新ERA的实际状态。因为实时使用的资源可能与所计划的不同，因此使用该方法很重要。例如，一些处理器可能失败或者掉线。最重要的是，我们可以预期，大多数作业实际上将使用比他们所预定的资源少得多的资源（为了确保资源足够，用户倾向于预定比实际需要更多的资源).ERA系统应当考虑这一因素，而且可能需要重新规划调度。

简单版本的云反馈包括以下信息：（1）云所管理的当前资源的变化（例如，某些计算机宕机）；（2）当前的资源消耗；（3）作业终止；（4）每个作业的等待进程数，这限定了此刻该作业可以使用的资源量（如果给该作业分配了无限资源的话）。

3. 算法

ERA的内部算法实现，采用了灵活的“即插即用”设计，封装在单独的组件（算法和预测组件）中，可以轻松切换算法以适应不同的条件和系统要求。算法组件执行作业请求的实际调度和定价。为了得到市场价格或需求预测，算法可以使用预测组件，而且ERA系统

会根据每个新请求在线更新预测组件。

3.1 调度和定价算法接口

ERA的算法实现了ERA系统的逻辑。ERA系统转发来自用户和云的查询，并调用算法应答，因此ERA和算法之间的内部接口类似于外部ERA接口（如第2.3.1节和第2.3.2节中所述），除了这个内部接口对所有与外部系统接口的复杂性进行了抽象。这两个接口（内部接口和外部接口）的主要不同是算法接口（即内部接口）没有给出预定请求的竞价，并且必须独立于竞价来定价。该接口只能对价值进行一次性比较，只要价值不低于价格，就会接受请求。因此，ERA架构强制算法在价值上必须是单调的（因此设定了获胜的阈值价格），从而创建了与价值相关的激励兼容机制；也就是说，由此设计的机制是真实可信的。

新作业的调度和定价由makeReservation方法执行。如第2.3.1节所述，该方法的输入是一个预定请求，形如“我需要W个核心，T个时间单位，在[开始时刻，截止时刻）之间的某个时间，最多愿意支付V的价格”。回应的形式是“接受/拒绝”，如果接受的话再加上一个价格P。ERA算法还应该追踪其所计划的分配，以便在云基础设施通过getCurrentAllocation方法查询的时候，告知其何时运行已接受的作业，并在更新查询时重新计划和优化（见第2.3.2节）。

基本经济调度

基本经济调度（算法1）是一个ERA算法的基础实现。新的作业请求到达时，算法会动态地为每个时间单位、每个资源单位（例如 秒*核心）设置一个价格 ，总价格就是所请求资源的这些单价之和。（如果是多种资源，将总价格简单泛化为累加各种资源的总价。因为简单和良好的经济特性（拆分请求并无助益），我们选择关注附加定价。）然后，算法会安排作业在窗口期内最便宜的时间启动(只要该作业的价值不低于计算出的总价格），以满足请求。为了确定未来特定时间单位t期间的资源价格，该算法考虑了已允诺的资源量和对该未来时段内需求的预测。本质上，根据预测的需求，算法设定的价格反映了因接受该作业而对未来请求施加的外部因素。需求预测封装在预测模组件中，将在下一节讨论。

注意，这个简单的算法放弃了抢占式作业（交换作业进出）的灵活性，而是以固定的开始时刻，为每个作业分配连续的时间间隔。它同时还分配请求的W个内核，而不是为了更少的并行内核而售卖更多时间。尽管基础实现中的这些灵活性是受ERA的应用程序接口支持的，但我们选择放弃，以便隔离所关注的每一个问题：这分割了算法问题（仅简单提及）和价格问题（真正要解决的问题），以及学习问题。另外，这样的调度在各种现实约束条件下是鲁棒的、可应用的，在其他情况下可能未到最优，可以作为基准。

Algorithm 1 Basic-Econ Scheduling

Input: a new job request {W*T in [A,D), V}
Output: accept or reject, and a price if accepted
procedure MAKE RESERVATION
    for each $t \in [A, D)$ **do**
        $demand_t() \leftarrow$ the demand estimate function at t
        for each $i \in [1, W]$ **do**
            $price_t(i) \leftarrow$ the highest price p s.t. $demand_t(p) + promised[t] + i > Capacity$
        $cost[t] \leftarrow price_t(1) + price_t(2) + ... + price_t(W)$
    for each $t \in [A, D - T]$ **do**
        $totalCost[t] \leftarrow cost[t] + ... + cost[t + T - 1]$
    $t^* \leftarrow \arg\min_{t \in [A, D-T]} totalCost[t]$
    if $V \geq totalCost[t^*]$ **then**
        schedule the job to start at t^*
        return accept at cost $totalCost[t^*]$
    else
        return reject
end procedure

3.2 需求预测接口

预测组件负责预测未来时间t在任意给定价格和给定当前时间的情况下的需求。由于我们真正需要的是反函数，我们的实际接口提供了这样的反函数：（然而，我们使用需求函数及其反函数来表示预测器。）给定未来时间t，当前时间和需求量q，反函数返回一个使得从当前时间到未来时间t的需求等于指定的需求量q的最高价格。

一般来说，我们无法指望可以确定未来的需求——因此，我们要求预测以最一般的形式指定售出资源数量关于价格的概率分布。由于这样的概率分布很难使用，所以我们的基础实现简化了预测问题，只要求预测器为每个需求量指定一个价格，这就像需求是确定型的。总需求是大量独立请求的聚合结果时，这种方法是合理的（译者按：大数定律）。在这种情况下，需求将是集中的（译者按：而非概率分布的），且单次预测的价格将合理地趋近价格分布。

基于数据的预测器：基于历史数据的预测

ERA的需求预测器根据作业请求的历史数据进行预估：以过去的请求列表作为输入，根据列表中的需求，在每个时间t学习需求曲线（需求作为价格的函数）。当然，这种方法带来了许多挑战：首先，存在“冷启动”问题——因为ERA为作业请求定义了一个新接口，

因此没有历史数据的情况ERA也能用这个新接口进行学习。其次，与预测的成功与否取决于确定需求周期的能力，例如一周的某一天或某几天。此外，学习方法还必须克服抽样误差并解决需求的不确定性（如上所述）。

我们的第一版实现没有处理这些挑战，而是旨在提出一种解决另一个重大挑战的方法：作业的时间灵活度。根本上说，问题在于我们期望预测器提供即时需求，而在ERA中，所请求的资源是一个时间窗口内（通常比其持续时间（译者按：作业运行时间）更长）的一段时间（译者按：即作业运行时间）的资源。因此，我们应该回答这个问题：一个作业请求在其请求的时间窗口内，如何影响已预测的需求？

一个本能的做法就是讲需求扩展到整个窗口期，例如，一个“10核心时长5分钟窗口期50分钟”的需求，扩展为10个“1核心时长5分钟窗口期50分钟”的需求。但这可能无法反映我们应该预期的实际需求。例如，考虑低中高价值作业的投入，每种作业都要求100%的容量，但是高价值的作业只能在白天运行，中低价值的作业白天晚上都能运行。用这种扩展的方式，我们在白天计算高价值作业的需求，在白天和夜晚分散中低价值的作业，这样我们就能在晚上获得50%的中低价值作业。这样给人的感觉是，我们用晚上的中等价值作业只能填充一般的产能，因此会定价太低，从而拒绝中等价值作业而接受低价值作业。

我们提出，这个问题可以通过采用基于LP松弛问题的方法来解决。基于LP的预测器离线运行一个线性程序，以查找以往的请求中最优的（即价值最大化的）部分分配，并用这个最优分配预测时间t的需求。请注意，LP需要许多变量——每个作业和作业时间窗口的每个时刻都需要一个变量，自由度可能非常巨大，因此人们怀疑在不同的时间所预测的需求差异很大，尽管这些不同时间的实际需求本质上是相同的。我们的初步实证测试表明，这种基于LP的方法是稳定的，不过还需要进一步测试，并且为其建立理论支撑。

4. ERA系统和仿真

ERA是一个完整的运行系统：我们将其实现为一个软件包，提供前文所描述的接口，包括定价、调度和预测算法的基础实现，这些算法都是可插拔的（译者按：松耦合的）且易于扩展或替换。此外，该系统包含一个仿真平台，可以模拟算法在给定作业请求的底层云模型的情况下的执行情况，其核心代码与和真实的云和真实的用户进行交互的代码是一致的。请参阅图1中的系统架构和图2中的仿真器屏幕截图。

我们在仿真器中多次运行ERA，以及两个云系统（Hadoop/YARN和微软的Azure Batch仿真器）的原型软件。接下来我们将展示一部分运行测试，来说明从简单的云定价系统（如现行使用的系统）向ERA（经济资源分配系统）转变所能带来的巨大潜在收益，并展示ERA与现有云系统进行整合的能力。

经济分配的重要性

我们首先通过考虑作业的价值，证明ERA系统能够显著提高云资源使用效率。我们使用仿真器输入作业，这些作业采样于雅虎的描述大规模MapReduce生产日志的分布[9]，并做了诸如加上截止时刻，和原始日志里没有涵盖的价值这样的修改。在这个输入中，有6类作业，每一类有大概1400到1500个作业。“雅虎-5”这一类作业的平均规模最大，我们将其设置为每单位1美元的较低平均价值，而为了对高价值生产作业进行建模，我们将其他所有类别的作业设为每单位10美元的高价值。（译者注：需要注意的是，）该集群非常小，不能满足所有的作业。

我们将ERA的基础经济调度算法与贪婪算法进行比较，贪婪算法没有考虑作业的价值，而是对每单位资源收取固定价格，并安排作业在要求的时间窗口内尽早运行。仿真结果表明，贪婪算法在大部分集群上运行了大量低价值作业（“雅虎-5”类），导致效益只有请求的总价值的10%。与此形成鲜明对比的是，ERA的基本经济算法能够了解作业的价值，并使用动态定价来接受更高价值的作业，效益达到了请求总价值的51%（请注意，达到100%是不可能的，因为云端集群太小而无法满足所有作业）。

*ERA-Rayon*的集成

我们接下来通过表明云系统成功地使用ERA运行真实作业，证明将ERA与真正的云系统集成是可行的。另外，我们表明ERA仿真器提供了一个很好的逼近真实执行的结果。

我们将ERA与Rayon[10]完全集成，这是一个处理计算资源预定的云系统，是Hadoop/YARN[31]（又叫MapReduce2.0）的一部分。集成工作首先要求我们在Rayon中引入经济考虑，因为Rayon的原始预定机制没有考虑预定的货币估值。接下来，我们通过添加一层简单的适配器类，将ERA的核心代码插入到Rayon的预定和调度过程中，这些适配器类在ERA和Rayon的应用程序接口之间建立了桥梁。桥接层通过getCurrentAllocation方法（参见第2.3.2节）配置Rayon，以完全遵照ERA的指令，但对该查询方法做了一个扩展：为目前在预定窗口期内却没有分配资源的作业添加了“空分配”（即分配的资源为零）。Rayon为ERA返回的每个作业开启一个队列，其中包括空分配的作业，因此Rayon能在可能的时候提前运行作业。

我们通过使用Gridmix平台生成的MapReduce作业的工作负载来测试集成的有效性。这些作业从100GB文件中读取和写入合成数据，一共处理了815份作业，全部顺利完成。作业请求在一小时的时间周期内抵达，平均要求3GB的内存，平均持续时间60秒（σ=6秒）。该集群由3个节点组成，每个节点80GB内存。Rayon的资源管理器配置为使用ERA和最简单的贪婪算法（如上所述）来分配单个资源——内存GB数（这个版本的Rayon只支持分配内

存）。

我们使用相同的贪婪算法在ERA仿真器中运行同样的作业工作负载，云模型每秒都会和ERA成功通信。这两次运行（在Rayon（Hadoop）系统和仿真器中）之间的比较显示，仿真器可以很好地近似云系统上的ERA的性能。我们发现，作业排定在大致类似的时间点上，且持续时间相似。两次运行的主要差别在于，仿真器在整个（仿真）执行过程中为作业分配了一个恒定容量，而真正的集群由于各种系统因素改变了容量，这脱离了ERA控制。在两次运行中，获得的总分配（GB*秒）类似：使用仿真的总分配为76730，而使用真实云系统的是77056.

测试*Azure Batch*

下一组仿真显示了在应用于云规模时使用ERA相较于现有算法的优势。在典型的云环境中，我们不能期望有一个ERA实例能够完全控制数百万个核心。因此，我们的目标是评估ERA是否能够与某个地区的核心子集一起工作，即使是在底层资源可用性不断变化的情况下。

仿真是在150000个内核组成的数据中心上进行的。ERA获得20%的资源，其他80%分配给非ERA请求，这些请求是使用标准Azure作业建模的。这意味着资源在基层地域不断被分配/释放，ERA必须予以考虑。ERA管理的资源中有20%来自可抢占资源，但并未限制只能单独使用这些可抢占资源。ERA本身作为Azure Batch仿真器的上层运行，该仿真器模拟微软Azure仿真器上的批处理工作负载。

ERA的基本经济调度算法相对于其他两种算法进行了实验：（1）按需算法：如果有足够的可用资源来启动和运行作业，则接受作业（可用性检查仅限于当前时刻，忽略持续时间内的资源可用性）。该算法安排所接受的作业立即运行，并收取固定价格；（2）贪婪（“先来先满足”）算法（如上所述），该算法收取非抢占资源价格的65%。

业内的普遍做法是限制非抢占资源的最大折扣。因此，在我们的实验中，ERA的基本经济调度算法收到限制，价格不会高于非抢占式作业，并且折扣不会超过35%。我们探索了经济算法的几种变体：（1）使用基于作业分布的先验知识的线性预测器，或使用过去观测的预测器；（2）对于以后的调度有或没有指数惩罚。每种变体都在使用算法的不同容量下进行测试，这样容量越高，作为重新运行已失败作业的储备资源就剩余越少。

仿真作业负载中的所有作业都请求一个以其请求时刻为开始时刻的时间窗口（即作业没有提前预定）。因为ERA获得了20%的资源，我们希望评估两个指标：（1）延迟作业比例：完成时刻晚于截止时刻的作业所占的百分比；（2）接受收入：因为我们只能对被接受的作业收取费用，算法越好，我们可以接受的作业就越多。图3表明ERA的经济算法在这两

个指标方面占优势。

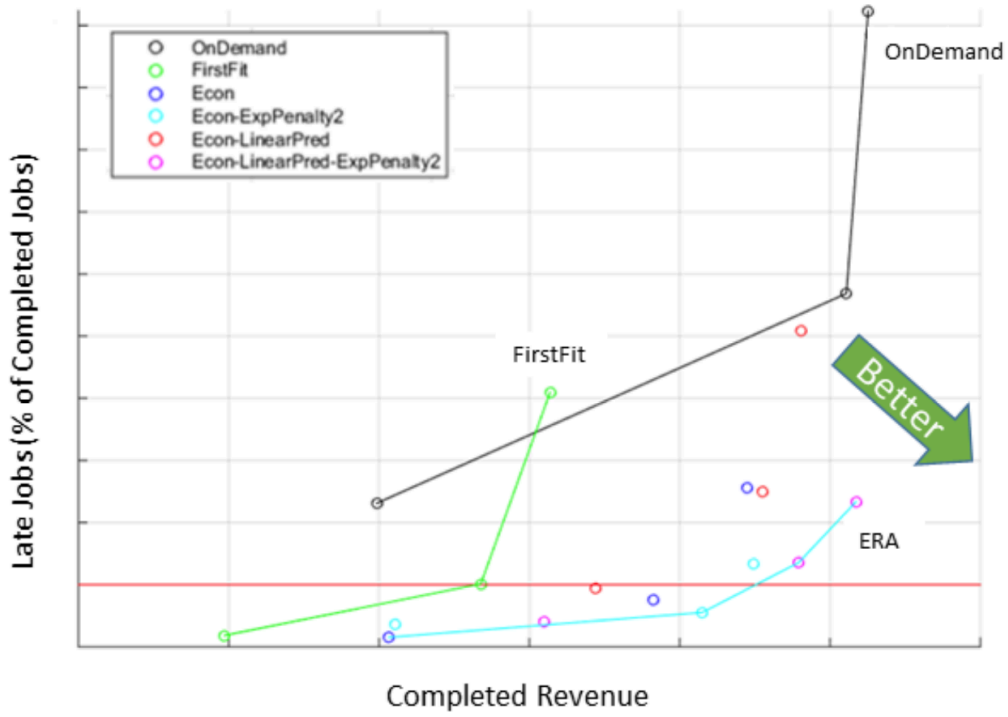

图3： ERA在Azure Batch和在仿真器上的结果（删除了轴比例）。依据二者期望的收入指标和延迟作业百分比。ERA的经济算法在需求和首次你和算法方面占据主导地位。

5. 巨大的挑战

5.1 作业调度

在随机模型和对抗模型中都有大量关于作业调度的文献。最明显相关的模型是具有松弛型的作业调度，所谓松弛，就是和有时间窗口的调度有所不同的地方。当前我们框架提出的问题在两个领域都会带来新的挑战。在我们的情况下，假定任何作业只需要总资源的一小部分是非常合理的，而松散度与作业规模相比非常大。一个有意思的现实问题是，让作业在时间（运行时间）和资源（机器数量）之间做出权衡，这取决于作业的并行化程度。另一个有意思的问题是，展示一个在随机模型和敌对模型之间进行插值的模型，其中随机模型提供了一个完整的作业到达过程模型，对抗模型则不做任何假设。如果有一个模型只需要几个参数且能捕获多个到达序列，那会非常好。最后，在随机模型和对抗模型中研究重复性的作业会非常有趣。

5.2 定价

在我们的模型中，我们假设用户既有明确的截止时刻，也有明确的运行时长。如果提

供更灵活的保障，能方便用户以更开放的方式进行偏好设定，这将会很有趣。例如，可以设定作业在一定的时间内以一定的成本和略低的优先级耗尽资源后继续运行额外的一段时间。另一个类似的保障是，用户可以设置“首选截止时刻”和“末选截止时刻”，这样大多数作业就能在首选截止时刻完成。所有这些都旨在为系统提供更为灵活的服务质量（QoS）保障。定价是一个非常重大的实践和理论挑战。

从理论上讲，为我们的系统提供理论依据会很好。首先，需要表明用户有动力真实地或者至少接近真实地汇报他们的信息，而不是试用和玩弄系统。其次，需要表明系统达到了可以满足要求的稳定状态（例如，显示合适的平衡状态和相关的无序价格）。

5.3 学习

我们提出的框架需要一个重要的学习组件。大部分学习依赖于过去观察到的时间序列，以用于预测未来的需求。在我们的设定中一个明显的挑战就是要适应季节性影响（每天地，白天对夜晚；每周地，工作日对周末；每年地，假期等）。这样的挑战在时序文献中非常普遍。更有趣的影响是，我们的系统的可用资源和需求都在不断增长，挑战就在于捆绑两个预测，或者又需要将其分开。似乎我们的预测模型需要一个比预期值更精确的预测，但对许多预测应用来说，我们需要得到更详细的信息。

另一个不确定性来自我们的系统可能无法知道某些请求——因为用户觉得这样的请求不太可能被接受，因此从未提交。例如，如果一项较重要的作业由于价值较低被拒绝，那么较不重要的作业可能就不会提交。所以，由于这些信息缺失，对需求的预测更具挑战性。

最后，学习不应仅限于需求预测，还应该预测请求的准确性。因为在目前的系统中我们要求作业不超出其最大长度（译者按：指作业的运行时长），因此估计可能是保守的，而学习什么是“实际”的需求，可能会释放大量资源。

5.4 鲁棒性

对于任何实际运行的系统来说，鲁棒性都非常重要。鲁棒性应该考虑到各种资源的各种计划的和意外的故障。对鲁棒性建模，可以作为服务质量保障这一更高挑战的一部分。我们应该研究可以提供什么样的极端情况保障。

参考文献

[1] Apache Hadoop Project. http://hadoop.apache.org/.
[2] V. Abhishek, I. A. Kash, and P. Key. Fixed and market pricing for cloud services. arXiv preprint arXiv:1201.5621, 2012.
[3] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing amazon

ec2 spot instance pricing. ACM Transactions on Economics and Computation, 1(3):16, 2013.
[4] Amazon. Amazon elastic mapreduce. At http://aws.amazon.com/elasticmapreduce/.
[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A view of cloud computing. CACM, 53(4):50–58, 2010.
[6] Y. Azar, I. Kalp-Shaltiel, B. Lucier, I. Menache, J. S. Naor, and J. Yaniv. Truthful online scheduling with commitments. In Proceedings of the Sixteenth ACM Conference on Economics and Computation, pages 715–732. ACM, 2015.
[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In ACM SIGOPS Operating Systems Review, volume 37, pages 164–177. ACM, 2003.
[8] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In OSDI, pages 285–300, Broomfield, CO, Oct. 2014. USENIX Association.
[9] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The case for evaluating mapreduce performance using workload suites. In Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, 2011.
[10] C. Curino, D. E. Difallah, C. Douglas, S. Krishnan, R. Ramakrishnan, and S. Rao. Reservation-based scheduling: If you're late don't blame us! In SoCC, 2014.
[11] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca. Jockey: guaranteed job latency in data parallel clusters. In Proceedings of the ACM European Conference on Computer Systems, EuroSys, 2012.
[12] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In NSDI, volume 11, pages 24–24, 2011.
[13] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. In ACM SIGCOMM Computer Communication Review, volume 44, pages 455–466. ACM, 2014.
[14] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. ACM SIGCOMM computer communication review, 2008.
[15] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: a platform for fine-grained resource sharing in the data center, 2011.
[16] N. Jain, I. Menache, J. S. Naor, and J. Yaniv. A truthful mechanism for value-based scheduling in cloud computing. Theory of Computing Systems, 54(3):388–406, 2014.
[17] S. A. Jyothi, C. Curino, I. Menache, S. M. Narayanamurthym, A. Tumanov,and et. al. Morpheus: Towards automated slos for enterprise clusters. In OSDI, 2016.
[18] K. Karanasos, S. Rao, C. Curino, C. Douglas, K. Chaliparambil, G. M. Fumarola, S. Heddaya, R. Ramakrishnan, and S. Sakalanaga. Mercury: Hybrid centralized and distributed scheduling in large shared clusters. In ATC, 2015.
[19] C. Kilcioglu and J. M. Rao. Competition on price and quality in cloud computing. In WWW, 2016.
[20] I. Menache, A. Ozdaglar, and N. Shimkin. Socially optimal pricing of cloud computing resources. In ICST Conference on Performance Evaluation Methodologies and Tools, 2011.
[21] I. Menache, O. Shamir, and N. Jain. On-demand, spot, or both: Dynamic resource allocation for executing batch jobs in the cloud. In 11th International Conference on Autonomic Computing (ICAC 14), pages 177–187, 2014.
[22] P. Menage, P. Jackson, and C. Lameter. Cgroups. Available on-line at: http://www.mjmwired.net/kernel/Documentation/cgroups.txt, 2008.
[23] D. Merkel. Docker: lightweight linux containers for consistent development and deployment. Linux Journal, 2014(239):2, 2014.
[24] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. Sparrow: Scalable scheduling for sub-second parallel jobs. Technical Report UCB/EECS-2013-29, EECS Department, University of California, Berkeley, Apr 2013.
[25] J. Rasley, K. Karanasos, S. Kandula, R. Fonseca, M. Vojnovic, and

S. Rao. Efficient queue management for cluster scheduling. In Proceedings
of the Eleventh European Conference on Computer Systems, page 36. ACM, 2016.
[26] D. Sarkar. Introducing hdinsight. In Pro Microsoft HDInsight, pages1–12. Springer, 2014.
[27] B. Sharma, R. K. Thulasiram, P. Thulasiraman, S. K. Garg, andR. Buyya. Pricing cloud compute commodities: a novel financial economic model. In IEEE-CCGRID, 2012.
[28] Y. Song, M. Zafer, and K.-W. Lee. Optimal bidding in spot instance market. In INFOCOM, 2012 Proceedings IEEE, pages 190–198.IEEE, 2012.
[29] M. A. team. Azure batch: Cloud-scale job scheduling and compute management. In https://azure.microsoft.com/en-us/services/batch/, 2015.
[30] A. Tumanov, T. Zhu, J. W. Park, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger. Tetrisched: global rescheduling with adaptive planahead in dynamic heterogeneous clusters. In Eurosys, 2016.
[31] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In ACM - SoCC, 2013.
[32] A. Velte and T. Velte. Microsoft virtualization with Hyper-V.
McGraw-Hill, Inc., 2009.
[33] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at google with borg. In Eurosys, 2015.
[34] C. A. Waldspurger. Memory resource management in vmware esx server. SOSP, 2002.
[35] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cl

参考文献原文

ERA: A Framework for Economic Resource Allocation for the Cloud

Moshe Babaioff[*] Yishay Mansour[†] Noam Nisan[‡] Gali Noti[‡] Carlo Curino[§] Nar Ganapathy[§] Ishai Menache[*] Omer Reingold[¶] Moshe Tennenholtz["] Erez Timnat["]

February 24, 2017

Abstract

Cloud computing has reached significant maturity from a systems perspective, but currently deployed solutions rely on rather basic economics mechanisms that yield suboptimal allocation of the costly hardware resources. In this paper we present Economic Resource Allocation (ERA), a complete framework for schedul- ing and pricing cloud

resources, aimed at increasing the efficiency of cloud resources usage by allocating resources according to economic principles. The ERA archi- tecture carefully abstracts the underlying cloud infrastructure, enabling the devel- opment of scheduling and pricing algorithms independently of the concrete lower- level cloud infrastructure and independently of its concerns. Specifically, ERA is designed as a flexible layer that can sit on top of any cloud system and interfaces with both the cloud resource manager and with the users who *reserve* resources to run their jobs. The jobs are scheduled based on prices that are *dynamically* calcu- lated according to the predicted demand. Additionally, ERA provides a key internal API to *pluggable* algorithmic modules that include scheduling, pricing and demand prediction. We provide a proof-of-concept software and demonstrate the effective- ness of the architecture by testing ERA over both public and private cloud systems
– Azure Batch of Microsoft and Hadoop/YARN. A broader intent of our work is to foster collaborations between economics and system communities. To that end, we have developed a simulation platform via which economics and system experts can test their algorithmic implementations.

[*] Microsoft Research, moshe,ishai@microsoft.com

[†] TAU and Microsoft Research, mansour@microsoft.com

[‡] HUJI and Microsoft Research, noam,gali.noti@cs.huji.ac.il

[§] Microsoft, ccurino,narg@microsoft.com

[¶] Stanford University, reingold@stanford.edu

["] Technion and Microsoft, moshet@ie.technion.ac.il, ereztimn@cs.technion.ac.il

1. Introduction

Cloud computing, in its private or public incarnations, is commonplace in industry as a paradigm to achieve high return on investments (ROI) by sharing massive computing infrastructures that are costly to build and operate [5,14]. Effective sharing pivots around two key ingredients: 1) a *system infrastructure* that can securely and efficiently multiplex a shared set of hardware resources among several tenants, and 2) *economic mechanisms* to arbitrate between conflicting resource demands from multiple tenants.

State-of-the-art cloud offerings provide solutions to both, but with varying degrees of sophistication. The system challenge has been subject to extensive research

focusing on space and time multiplexing. Space multiplexing consists of sharing servers among tenants, while securing them via virtual machine [34,32,7] and container technologies [23,22]. Time multiplexing comprises a collection of techniques and systems that sched- ule tasks over time. The focus ranges from strict support of Service Level Objectives (SLOs) [10,17,30,11] to maximization of cluster utilization [12,35,18,25,24,13,8].

Many of these advances are already deployed solutions in the public cloud [26,4] and the private cloud [1,31,15,33,8]. This indicates a good degree of maturity in how the system challenge is tackled in cloud settings.

On the other hand, while the economics challenge has received some attention in re- cent research (see, e.g., [27,16,6,28,19,20] and references therein), the underlying principles have not yet been translated into equally capable solutions deployed in prac- tice.

1.1. The Economic Challenge and ERA's Approach

In current cloud environments, resource allocation is governed by very basic economics mechanisms. The first type of mechanism (common in private clouds [31,8,15,33]) uses fixed pre-paid guaranteed quotas. The second type (common in public clouds [26, 4]) uses on-demand unit prices: the users are charged real money[1] per unit of resource used. In most cases these are fixed prices, with the notable exception of Amazon's spot instances that use dynamically changing prices.[2] Spot-instance offerings, however, do not provide guaranteed service, as the instances might be evicted if the user bid is too low. Hence, utilizing spot instances requires special attention from the user when determining his bid, and might not be suitable for high-priority production jobs [21,28,2]. The fundamental problem is finding a pricing and a scheduling scheme that will result in highly desired outcome, that of high efficiency.

Efficiency: From an economics point of view, the most fundamental goal for a

---

cloud system is to maximize the economic *efficiency*, that is, to maximize the total value that

[1] Or, within a company, fiat money.

[2] Although, based on independent analysis, even these may not truly leverage market mechanisms to determine prices [3].

all users get from the system. For example, whenever two users have conflicting de- mands,the one with the lowest cost for switching to an alternative (running at a different time/place or not running at all) should be the one switching. The resources would thus be allocated to the user with "highest marginal value."The optimal- allocation bench- mark for a given cloud is that of an omniscient scheduler who has access to the complete information of all cloud users – including their internal costs and alternative options – and decides what resources to allocate to whom in a way that maximizes the efficiency goals of the owner of the cloud. Let us stress: to get a meaningful measure of efficiency we must count the *value obtained* rather than the *resources used*, and we should aim to maximize this value-based notion of efficiency.[3]

Limitations of current solutions: With this value-based notion of efficiency in mind, let us evaluate commonly deployed pricing mechanisms. Private cloud frameworks [31, 8,15,33] typically resort to pre-paid guaranteed quotas. The main problem with this option is that, formally, it provides no real sharing of common resources: to guarantee that every user always has his guaranteed pre-paid resources available, the cloud sys- tem must actually hold sufficient resources to satisfy the sum of all promised capacities, even though only a fraction will likely be used at any given time.Mechanisms such as work-preserving fair-queueing [12] are typically designed to increase utilization [31,15], but they do not fundamentally change the equation for value-based efficiency, as the re- sources offered above the user quota are typically distributed at no cost and with no guarantees.Furthermore, lump-sum pre-payment implies that the users' marginal cost for using their guaranteed resources is essentially zero, and so they will tend to use their capacity for "non-useful" jobs whenever they do not fill their capacity with "useful" jobs. This often results in cloud systems that seem to be operating at full capacity from ev- ery engineering point of view, but are really working at very "low capacity" from an economics point of view, as most of the time, most of the jobs have very low value.

On the other hand, public cloud offerings [26,4] typically employ on-demand unit-pricing schemes.The issue with this solution is that the availability of resources cannot be guaranteed in advance. Typically the demand is quite spiky, with short periods of peak demand interspersed within much longer periods of low demand. Such spiky demand is also typical of many other types of shared infrastructure such as computer network bandwidth, electricity, or ski resorts. In all these cases the provider of the shared resource faces a dilemma between extremely expensive over-provisioning of capacity and giving up on guaranteed service at peak times. In the case of cloud systems, for jobs that are important enough, users cannot take the risk of their jobs not being able to run when needed, and thus "on-demand" pricing is only appealing to low-value or highly time-flexible jobs, while most important "production" jobs with little time flexibility resort to buying long-term guaranteed access to resources. While flexible unit prices (such as Amazon's spot instances) may have an advantage over fixed ones as they can better smooth demand over time, as highlighted above, they only get an opportunity to do so for the typically low-value "non-production" jobs.

[3] Another important goal, of course, is revenue, but we note that the potential revenue is limited by the value created, so high efficiency is needed for high revenue. Moreover, since there is competition between cloud providers, these providers generally aim to increase short-term efficiency as this is likely to have positive long-term revenue effects. The issue of increasing revenue is usually attacked under the "Platform as a Service" (PaaS) strategy of providing higher-level services. This is essentially orthogonal to allocation efficiency and is beyond the scope of the present paper.

The ERA approach: The pricing model that we present in ERA enables sharing of re- sources and smoothing of demand even for high-value production jobs. This is done using the well-known notion of *reservations*, commonly used for many types of resources such as hotel rooms or super-computer time as well as in a few cloud systems [10,17,30], but in a *flexible way in terms of both pricing and scheduling.* We focus on the economic chal- lenges of scheduling and pricing batch style computations with completion-time SLOs (deadlines) on a shared cloud infrastructure. The basic model presented to the user is that of *resource reservation*. At "reservation time," the user's program specifies its reservation request. The basic form of such a request is:[4]

Basic Reservation: *"I need 100 containers (with 6GB and 2cores each) for 5 hours, some time between 6am and 6pm today, and am willing to pay up to $100 for it."*

This class of workloads is very prominent – much of "big data" falls under this cat- egory [11,30,17] – and it provides us with a unique opportunity for economic inves- tigation. While state-of-the-art solutions provide effective system-level mechanisms for sharing resources, they rely on users' goodwill to truthfully declare their resource needs and deadlines to the system. By dynamically manipulating the price of resources, ERA provides users with incentives to expose to the system as much flexibility as possible. The ERA mechanism ensures that the final price paid by the user is the *lowest possible price* the system can provide for granting the request. The more flexibility a user ex- poses, the better the user's chances of getting a good price. If this minimal price exceeds the stated maximal willingness to pay, then the request is denied.[5] Once a reservation request is accepted, the payment is fixed at reservation time, and the user is assured that the reserved resources will be available to him within the requested window of time. The guarantee is to satisfy the request rather than provide a promise of specific resources at specific times. For more details regarding the model presented to the user see Section 2.3.1.

[4]The general form of requests is given by ERA's "bidding description language" (see Section2.3.1) that allows specifying multiple resources, variable "shapes" of use across time, and combinations thereof. [5]Alternatively, the ERA mechanism may present this minimal price as a "quote" to the user, who may
decide whether to accept or reject it.

1.2. An Overview of ERA

A key part of the challenge of devising good allocation schemes for cloud resources is their multi-faceted nature: while our goals are in terms of economic high-level business considerations, implementation must be directly carried out at the computer systems- engineering level. These two extreme points of view must be connected using clever algorithms and implemented using appropriate software engineering.

Indeed, in the liter- ature regarding cloud systems, one can see many papers that deal with each one of these aspects – "systems" papers as well as theoretical papers on scheduling and pricing jobs in cloud systems – as cited above. Unfortunately, these different threads in the literature are often disconnected from each other and they are not easy to combine to get an overall so- lution. We believe that one key challenge at this point is to provide a common abstraction that encompasses all these considerations. We call this the *architectural challenge*.

Our answer to this challenge is the *ERA system* (Economic Resource Allocation). The ERA system is designed as an intermediate layer between the cloud users and the underlying cloud infrastructure. It provides a single model that encompasses all the very different key issues underlying cloud systems: economic, algorithmic, systems-level, and human-interface ones. It is designed to integrate economics insights practically in real- world system infrastructures, by guiding the resource allocation decisions of a cloud sys- tem in an economically principled way.[6] This is achieved by means of a key architectural abstraction: *the ERA Cloud Interface*, which hides many of the details of the resource management infrastructure, allowing ERA to tackle the economic challenge almost inde- pendently of the underlying cloud infrastructure. ERA satisfies three key design goals:
1) it provides a crisp theoretical abstraction that enables more formal studies; 2) it is a practical end-to-end software system; and 3) it is designed for extensibility, where all the algorithms are by design easy to evolve or experiment with.

ERA's key APIs: ERA has two main outward-facing APIs as well as a key internal API. Figure1gives a high-level schematic of the architecture. The first external API faces the users and provides them with the economic reservation model of cloud services described above. The second external API faces the low-level cloud resource manager. It provides a separation of concerns that frees the underlying cloud system from any time-dependent scheduling or from any pricing concerns, and frees the ERA system from the burden of assigning specific processors to specific tasks in a reasonable resource-locality way, or from the low-level mechanics of firing up processes or swapping them out. See more details in Section2.3.

Finally, the internal API is to pluggable algorithmic scheduling, pricing, and

predic- tion modules. Our basic scheduling and pricing algorithm dynamically computes future resource prices based on supply and demand, where the demand includes both resources

[6]Not all resources in the cloud have to be managed via ERA. It is also possible that the cloud will let ERA manage only a subset of the resources (allowing the system to be incrementally tested), or will have several instances of ERA to manage different subsets of the resources.

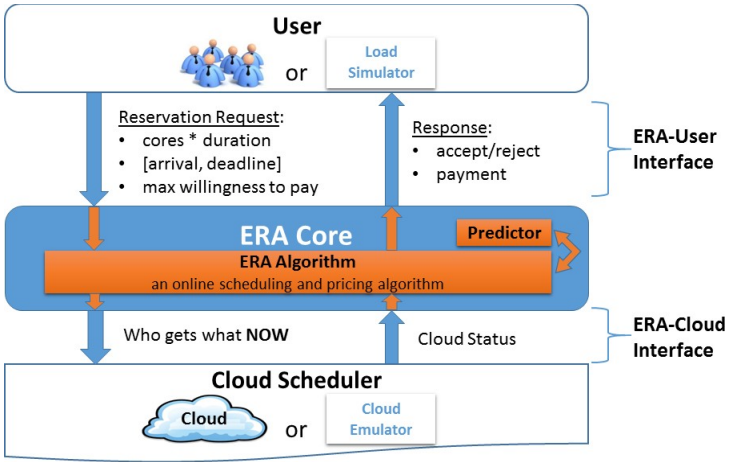

Figure 1: ERA Architecture. The ERA system is designed as an intermediate layer between the users and the underlying cloud infrastructure. The same actual core code is also interfaced with the simulator components.

that are already committed to and predicted future requests, and schedules and prices the current request at the "cheapest" possibility. Our basic prediction model uses traces of previous runs to estimate future demand. The flexible algorithmic API then allows for future algorithmic, learning, and economic optimizations. The internal interfaces as well as our basic algorithmic implementations are described in Section 3.

Our goal in defining this abstraction is more ambitious than mere good software en- gineering in our system. As part of the goal of fostering a convergence between system and economic considerations, we have also built a flexible cloud simulation framework. The simulator provides an evaluation of key metrics, both "system ones"

such as loads or latency, as well as “economic ones” such as “welfare” or revenue, as well as provides a visualization of the results (see screenshot in Figure2). The simulator was designed to provide a convenient tool both for the cloud system’s manager who is interested in eval- uating ERA’s performance as a step toward integration and for researchers who develop new algorithms for ERA and are interested in experimenting with their implementation without the need to run a large cluster. As is illustrated in Figure1, the same core code that receives actual user requests and runs over the underlying cloud resource manager may be connected instead to the simulator so as to test it under variable loads and alter- native cloud models. Comparing the results from our simulator and physical cluster runs, we find the simulator to be faithful (Section4).

The ERA system is implemented in Java, and an alternative implementation (of a subset of ERA) in C# was also done. We have performed extensive runs of ERA within

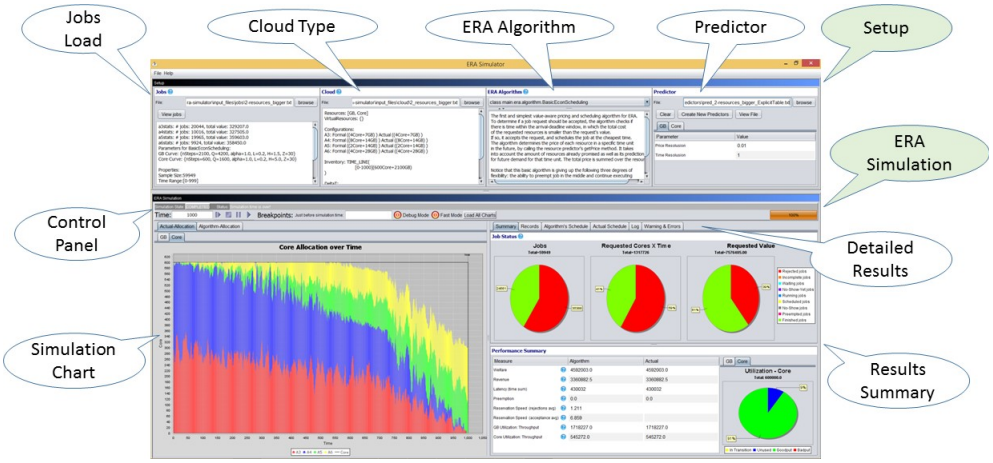

Figure 2: ERA Simulator Screenshot

the simulator as well as proof-of-concept runs with two prominent resource managers in the public and private clouds: the full system was interfaced with Hadoop/ YARN

31. and the C# version of the code was interfaced and tested with Microsoft’s Azure Batch[7] simulator [29]. These runs show that the ERA algorithms succeed in increasing the efficiency of cloud usage, and that ERA can be successfully

integrated with real cloud systems. Additionally, we show that the ERA simulator gives a good approximation to the actual run on a cloud system and thus can be a useful tool for developing and testing new algorithms. In Section4we present the results of a few of these runs.

Contributions: In summary, we present ERA, a reservation system for pricing and scheduling jobs with completion-time SLOs. ERA makes the following contributions:

1. We propose an abstraction and a system architecture that allows us to tackle the economic challenge orthogonally to the underlying cloud infrastructure.
2. We devise algorithms for scheduling and pricing batch jobs with SLOs, and for predicting resource demands.
3. We design a faithful cloud simulator via which economics and system experts can study and test their algorithmic implementations.
4. We integrate ERA with two cloud infrastructures and demonstrate its effectiveness experimentally.

[7] Azure Batch is a cloud-scale job-scheduling and compute management service. https://azure.microsoft.com/en-us/services/batch/

2. The ERA Model and Architecture

2.1. The Bidding Reservation Model with Dynamic Prices

ERA is designed to handle a set of computational resources of a cloud system, such as cores and memory, with the goal of allocating these resources to users efficiently. The basic idea is that a user that needs to run a job at some future point in time can make a reservation for the required resources and, once the reservation is accepted, these resources are then guaranteed (insofar as physically possible) to be available at the reserved time. The guarantee of availability of reserved resources allows high-value jobs to use cloud just like in the pre-paid guaranteed quotas model, but without the

need to buy the whole capacity (for all times), which thus also allows for time sharing of resources, which increases efficiency.

The price for these resources is quoted at reservation time and is dynamically com- puted according to (algorithmically estimated) demand and the changing supply. More user flexibility in timing is automatically rewarded by lower prices. The basic idea is that these dynamic prices will regulate demand, achieving a better utilization of cloud re- sources. This mechanism also ensures that at peak times – where demand can simply not be met – the most "valuable" jobs are the ones that will be run rather than arbitrary ones. ERA uses a simple bidding model in which the user attaches to each of his job requests a monetary value specifying the maximal amount he is willing to pay for running the job. The amount of value lost for jobs that cannot be accommodated at these peak times serves as a quantification of the value that will be gained by buying additional cloud resources, and is an important input to the cloud provider's purchasing decision process.

2.2. The Cloud Model

The cloud in the ERA framework is modeled as an entity that sells multiple resources, bundled in configurations, and the capacity of these resources may change over time. The configurations, as well as the new concept of "virtual resources," are designed to represent constraints that the cloud is facing, such as packing constraints. Specifically, the cloud is defined by: (1) a set of formal resources for sale (e.g., core or GB). We also allow for capturing additional constraints of the underlying infrastructure by using a notion of "virtual resources"; (2) a set of resource configurations: each configuration is defined by a bundle of formal resources (e.g., "ConfA" equals 4 cores and 7 GB),[8] and is also associated with a bundle of actual resources that reflects the average amount the system needs in order to supply the configuration. The actual resources will typically be larger than the formal resources. The gap is supposed to model the overhead the

[8] These configurations are preset, but notice that ERA's cloud model also supports the flexibility that

each job can pick its own bundle of resources (as in YARN) by defining configurations of the basic formal resources (e.g., “ConfCore” equals a single core).

cloud incurs when trying to allocate the formal amount of resources within a complex system. The actual resources can be composed of formal as well as virtual resources; (3) inventory: the amount of resources (formal and virtual) over time; (4) time definitions of the cloud (e.g., the precision of time units that the cloud considers).

2.3. The ERA Architecture

The ERA system is designed as a smart layer that lies between the user and the cloud scheduler, as shown in Figure1. The system receives a stream of job reservation requests for resources arriving online from users. Each request describes the resources the user wishes to reserve and the time frame in which these resources are desired, as well as an economic value specifying the maximal price the user is willing to pay for these re- sources. ERA grants a subset of these requests with the aim of maximizing total value (and/or revenue). The interface with these user requests is described in Section2.3.1.

ERA interfaces with the cloud scheduler to make sure that the reservations that were granted actually get the resources they were promised. ERA instructs the cloud how it should allocate its resources to jobs, and the cloud should be able to follow ERA’s instructions and (optionally) to provide updates about its internal state (e.g., the capacity of available resources), allowing ERA to re-plan and optimize. ERA’s interface with the cloud scheduler is described in Section2.3.2.

The architecture encapsulates the logic of the scheduling and pricing in the *algorithm* module. The algorithms use the *prediction* module to compute prices dynamically based on the anticipated demand and supply. This architecture gives the ability to change be- tween algorithms and to apply different learning methods. ERA’s internal interface with the algorithmic components is described in Section3.

2.3.1. ERA-User Interface

The ERA-User interface handles a stream of reservation requests that arrive online from users, and determines which request is accepted and at which price.

The Bidding Description Language Each reservation request bids for resources according to ERA's bidding description language – an extension of the reservation defini- tion language formally defined in [10]. The bid is composed of a list of *resource requests* and a maximum willingness to pay for the whole list. Each resource request specifies the configurations of resources that are requested, the length of time for which these are needed, and a time window $[arrival, deadline)$. All the resources must be allocated after the arrival time (included) and before the deadline (excluded). For example, a resource request may ask for a bundle of 3 units of ConfA and 2 units of ConfB, for a duration of 2 hours, sometime between 6AM and 6PM today. Each configuration is composed of one or more resources, as described in Section2.2.

By supporting a list of resource requests, ERA allows the description of more com- plex jobs, including the ability to break each request down to the basic resource units al- lowing for MapReduce kinds of jobs, or to specify order on the requests to some degree. The current ERA algorithms accept a job only if all of the resource requests in the list can be supplied; i.e., they apply the AND operator between resource requests. More so- phisticated versions may allow more complex and rich bidding descriptions, e.g., support of other operators or recursive bids. For clarity of presentation, in this paper we present ERA in the simple case, where the reservation request is a single resource request, and there is only a single resource rather than configurations of multiple resources.

The *makeReservation* method The interface with the user reservation requests is com- posed of the single "*makeReservation*" method, which handles a job reservation request that is sent by some user. Each reservation request can either be accepted and priced or declined by ERA.

The basic input parameters to this method are the job's bid and the identifier of the job. The bid encapsulates a list of resource requests along with the maximum price

that the user is willing to pay in order to get this request (as described above). The output is an acceptance or rejection of the request, and the price that the user will be charged for fulfilling his request in case of acceptance.[9]

The main effect of accepting a job request is that the user is guaranteed to be given the desired amount of resources sometime within the desired time window. An accepted job must be ready to use all requested resources starting at the beginning of the requested window, and the request is considered fulfilled as long as ERA provides the requested resources within the time window.

2.3.2. ERA-Cloud Interface

The interface between ERA and the cloud-scheduler is composed of two main methods that allow the cloud to get information about the allocation of resources it should apply at the current time, and to provide ERA with feedback regarding the actual execution of jobs and changes in the cloud resources.

The *getCurrentAllocation* method This is the main interface with the actual cloud scheduler. The cloud should repeatedly call this method (quite often, say, every few seconds) and ask ERA for the current allocations to be made.[10] The method returns an *allocation*, which is the list of jobs that should be instantaneously allocated resources and

---

[9] Alternatively, the system may allow determining the payment after running is completed (depending on the system load at that time), or may allow flexible payments that take into account both the amount of resources reserved and the amount of resources actually used.

[10] For performance, it is also possible to replace this query with an event-driven scheme in which ERA pushes an event to the cloud scheduler when the allocations change.

the resources that should be allocated to them. In the simple case of a single resource, it is a list of "job J should now be getting W resources." The actual cloud infrastructure should update the resources that it currently allocates to all jobs to fit

the results of the current allocation returned by this query. This new allocation remains in effect until a future query returns a different allocation. It is the responsibility of the underlying cloud scheduling system to query ERA often enough, and to put these new allocations into effect ASAP, so that any changes are effected with reasonably small delay. The main responsibility of the ERA system is to ensure that the sequence of answers to this query reflects a plan that can accommodate all accepted reservation requests.

The main architectural aspect of this query is to make the interface between ERA and the cloud system narrow, such that it completely hides the plan ERA has for future allocation. It is assumed that the cloud has no information on the total requirements of the jobs, and follows ERA as accurately as possible.

The *update* method (optional usage) The cloud may use this method to periodically update ERA with its actual state. Using this method is important since the way resources are actually used in real time may be different from what was planned for. For example, some processors may fail or be taken offline. Most importantly, it is expected that most jobs will use significantly less resources than what they reserved (since by trying to en- sure that they have enough resources to usually complete execution, they will probably reserve more than they actually use). The ERA system should take this into account and perhaps re-plan.

The simple version of the cloud feedback includes: (1) changes in the current resources under the cloud's management (e.g., if some computers crashed); (2) the current resource consumption; (3) termination of jobs; (4) the number of waiting processes of each job, which specifies how many resources the job could use at this moment, if the job were allocated an infinite amount.

3. Algorithms

The internal algorithmic implementation of ERA is encapsulated in separate components

– the *algorithm* and the *prediction* components – in a flexible "plug and play" design, allowing to easily change between different implementations to fit different conditions and system requirements. The algorithm component is where the actual scheduling and pricing of job requests are performed. The algorithm may use the prediction component in order to get the market prices or the estimated demand, and the ERA system updates the prediction component online with every new request.

3.1. Scheduling and Pricing Algorithms

Interface The ERA algorithm is an online scheduling and pricing algorithm that pro- vides the logic of an ERA system. The ERA system forwards queries arriving from users and from the cloud to be answered by the algorithm, and so the internal interface between ERA and the algorithm is similar to the external ERA interface (described in Sections2.3.1and2.3.2), except that it abstracts away all the complexities of interfacing with the external system. The main change between these two interfaces is that the al- gorithm is not given the bids (the monetary value) of the reservation requests, and must decide on the price *independently* of the bid. It can only make a one-time comparison against the value, and the request is accepted as long as the value is not smaller than the price. Thus, the architecture enforces that the algorithm will be monotonic in value (as it sets a threshold price for winning), creating an incentive-compatible mechanism with respect to the value; i.e., the resulting mechanism is *truthful by design*.

The scheduling and pricing of a new job is performed in the *makeReservation* method. As described in detail in Section2.3.1, the input to this method is a reservation re- quest of the form "I need W cores for T time units, somewhere in the time range [Ar- rival,Deadline), and will pay at most V for it." The answers are of the form "accept/reject" and a price P in case of acceptance. The algorithm should also keep track of its planned allocations to actually tell the cloud infrastructure when to run the accepted jobs upon a *getCurrentAllocation* query, and re-plan and optimize upon an *update* query (see Section 2.3.2).

Basic Econ Scheduling The *Basic Econ Scheduling* (Algorithm1) is our basic implementation of an ERA algorithm. Whenever a new job request arrives, the algorithm dynamically sets a price for each time and each unit of the resource (e.g., second*core), and the total price is the sum over these unit prices for the requested resources.[11] It then schedules the job to start at the cheapest time within its requested window that fits the request, as long as the job's value is not lower than the computed total price. To deter- mine the price of a resource in a specific time unit t in the future, the algorithm takes into account the amount of resources already promised as well as its prediction for future demand for that time unit. Essentially, the price is set to reflect the externalities imposed on future requests due to accepting this job, according to the predicted demand. The prediction of demand is encapsulated in the prediction component we will discuss in the next section.

Note that this simple algorithm gives up the flexibility to preempt jobs (swap jobs in and out) and instead allocates to each job a continuous interval of time with a fixed starting time. It also allocates exactly the W requested cores concurrently instead of

[11] In case of multiple resources, the simple generalization is to set the total price additively over the different types of resources. We choose to focus on additive pricing due to its simplicity and good economic properties (e.g., splitting a request is never beneficial).

trading off more time for less parallel cores. We chose to give up these flexibilities in the basic implementation, although they are supported by the ERA API, in order to isolate concerns: this choice separates the algorithmic issues (which are attacked only in a basic way) from pricing issues (which are dealt with) and from learning issues. In addition, such schedules are robust and applicable under various real-world constraints, and in other cases they may simply be suboptimal and serve as benchmarks.

Algorithm 1 Basic-Econ Scheduling

1: **Input**: a new job request W(*T in [A,D), V)

2: **Output**: accept or reject, and a price if accepted

3: **procedure** MAKE RESERVATION

4: **for each** ∈ [*A, D*) **do**

5: $demand_t(\cdot) \leftarrow$ the demand estimate function at t

6: **for each** ∈ [1, *W*] **do**

7: ← $price_t(i)$ the highest price p s.t. $demand_t(p) + promised[t] + i > Capacity$

8: $cost[t] \leftarrow price_t(1) + price_t(2) + ... + price_t(W)$

9: **for each** $t \in [A, D - T]$ **do**

10: $totalCost[t] \leftarrow cost[t] + ... + cost[t + T - 1]$

11: $t^* \leftarrow \arg\min_{t \in [A,D-T]} totalCost[t]$

12: **if** $V \geq totalCost[t^*]$ **then**

13: schedule the job to start at t^*

14: **return** accept at cost $totalCost[t^*]$

15: **else**

16: **return** reject

17: **end procedure**

3.2. Demand Prediction

Interface The prediction component is responsible for providing an estimation of demand at a future time t at any given price, given the current time. Since the inverse function is what we really need, our actual interface provides that inverse function:[12] given a future time t, the current time, and a quantity of demand q, it returns the highest price such that the demand that arrives from the current time till t, at this price, is equal to the specified quantity q.

In general, one cannot expect future demand to be determined deterministically –

thus a prediction would, in its most general form, be required to specify a probability

[12]Yet, we present the predictors using both the demand function and its inverse. Moving between the two is straightforward.

distribution over prices that will result in selling the specified quantity. As such an object would be hard to work with, our basic implementation simplifies the prediction problem, and requires the predictor to only specify a single price for each demand quantity, as if demand is deterministic. Such an approach is justified when the total demand is a result of the aggregation of a large number of independent requests. In that case the demand will be concentrated and the single expected price will reasonably approximate the price distribution.

Data-based predictors: prediction based on historic data ERA's predictor – the demand oracle – builds its estimations based on historic data of job requests. It gets as input a list of past requests, and learns, for every time t, the demand curves (i.e., the demand as a function of price) according to the demand in the list. Of course, this approach presents multiple challenges: first, there is the "cold start" problem – as ERA defines a new in- terface for job requests, there are no past requests of the form that ERA can use to learn. Second, the success of the prediction depends on the ability to determine cycles in the demand, such as day-night or days of the week. In addition, the learning methods must also overcome sampling errors and address the non-deterministic nature of the demand (as discussed above).

Our first implementation of a data-based predictor puts these challenges aside and aims to suggest an approach to address an additional major challenge: the time flexibility of jobs. Essentially, the problem is that we expect the predictor to provide the instan- taneous demand, while in ERA the requests are for resources for some duration, within a window of time that is usually longer than the duration. Thus, we should answer the following question: how should a job request affect the predicted demand in its requested time window?

The naive approach would be to spread the demand over the window, e.g., a request of 10 cores for 5 minutes over a window of 50 minutes would contribute 1 core demand in each of the 50 minute window. However this may not reflect the actual

demand we should expect. For example, consider the input of low-, medium-, and high-value jobs. Each type asks for 100% of the capacity, where the high-value jobs can run only during the day and the low- and the medium-value jobs can run either day or night. Using the spreading approach we count the demand of the high-value jobs at day, and spread the low- and medium-value jobs over day and night, such that at night we obtain a demand of 50% of the low- and 50% of the medium-value jobs. Using this demand gives the impression that we can fill only half of the capacity using the medium-value jobs at night, and so we will set the price to be too low, and will accept low-value jobs at the cost of declining medium-value ones.

We suggest that this problem can be overcome by taking a different approach based on the LP relaxation of the problem. The *LP-based predictor* runs a linear program, offline, to find the optimal (value-maximizing) fractional allocation over past requests,

and predicts the demand at time t using the fractional optimal allocation at that time. Note that this LP requires many variables – one variable for every job and every time in the job's time window, and the number of degrees of freedom may be large, and so one may suspect that the predicted demand will be very different at different times that are experiencing essentially the same demand. Our preliminary empirical tests suggest that this LP-based approach is stable, yet future work should test this further and establish theoretical justifications for the approach.

4. The ERA System and Simulations

ERA is a complete working system: it is implemented as a software package that provides the interfaces described above together with basic implementations of the pricing, scheduling, and prediction algorithms, which are pluggable and can be extended or re- placed. In addition, the system contains a simulation platform that can simulate the execution of an algorithm given a sequence of job requests and a model of the underlying cloud, using exactly the same core code that is interfaced with the real cloud and users. See the system architecture in Figure1and a screen-shot of the

simulator in Figure2.

We have performed extensive runs of ERA within the simulator as well as proof-of-concept runs with two cloud systems: Hadoop/ YARN and Microsoft's Azure Batch simulator. Next we present a few of these runs to demonstrate the large potential gains of moving from the simple cloud-pricing systems, like the ones currently in use, to ERA

– the Economic Resource Allocation system – and to demonstrate the ability of the ERA system to integrate with existing cloud systems.

The importance of economic allocation We first demonstrate the ability of the ERA system to improve the efficiency of cloud resource usage significantly, by considering the jobs' values. We use the simulator with input of jobs that were sampled according to distributions describing a large-scale MapReduce production trace at Yahoo [9], after some needed modifications of adding deadlines and values that were not included in the original trace. In this input, there are 6 classes of jobs, and about 1,400–1,500 jobs of each class. Jobs of class "yahoo-5" have the largest average size, and we set them to have a low average value per unit of $1, while we set jobs of all other classes to have a high value per unit of $10, to model high-value production jobs. The cluster is way too small to fit all jobs.

We compare ERA's Basic-Econ scheduling algorithm with a gre-edy algorithm that does not take into account the values of the jobs, but instead charges a fixed price per resource unit, and that schedules the job to run at the earliest possible time within its requested time window. The simulation shows that the greedy algorithm populates most of the cluster with the large, low-value jobs (of class yahoo-5) and results in a low ef- ficiency of only 10% of the total requested value. In sharp contrast, ERA's Basic-Econalgorithm, which is aware of the values of the jobs and uses dynamic pricing to accept the higher-value jobs, achieves 51% of the requested value (note that getting 100% is not possible as the cloud is too small to fit all jobs).

ERA–Rayon integration We next demonstrate that it is feasible to integrate ERA with a real cloud system by showing that the cloud succeeds in running real jobs using

ERA. In addition, we show that the ERA simulator provides a good approximation to the outcome of the real execution.

We have fully integrated ERA with Rayon [10], which is a cloud system that handles reservations for computational resources, and is part of Hadoop/YARN [31] (aka MapRe- duce 2.0). The integration required, first, that we introduce economic considerations into the Rayon system, as Rayon's original reservation mechanism did not consider the reser- vations' monetary valuations. Next, we plugged ERA's core code into Rayon's reser- vation and scheduling process, by adding a layer of simple adapter classes that bridge between ERA's and Rayon's APIs. The bridging layer configured Rayon to completely follow ERA's instructions via the *getCurrentAllocation* method (see Section2.3.2), but made one extension to this query: it added an "empty allocation" (i.e., allocation of zero resources), for jobs that are during their reservation time-window but are currently not allocated resources. Rayon opened a queue for each job that was returned by ERA, in- cluding jobs with an empty allocation, and thus it was able to run jobs earlier than they were scheduled when it was possible.

We tested the integration by using a workload of MapReduce jobs that we generated using the Gridmix[13] platform. The jobs read and wrote synthetic data from files of 100 GB created for this purpose. Eight hundred and fifteen jobs were processed, all of which finished successfully. They arrived during a period of one hour, asked on average for 3 GB memory, for a duration of 60 seconds on average (σ = 6 seconds). The cluster consisted of 3 nodes, of 80 GB memory each. Rayon's resource manager was configured to use ERA with the simplest greedy algorithm (described above) that allocates a single resource – GB of memory (as the version of Rayon at the time allocated only memory).

We ran the same job workload in the ERA simulator, with the same greedy algo- rithm, and a cloud model that communicates with ERA every second with no failures. The comparison between these two runs – over Rayon (Hadoop) system and in the simu- lator – shows that the simulator gives a good approximation to the performance of ERA on a cloud system. We found that jobs were scheduled and running on approximately similar points in time and had similar durations. The main difference

between the two runs is that while the simulator assigns jobs a constant capacity throughout their (sim- ulated) execution, the real cluster changes their capacity according to various system considerations that are out of ERA's control. The total allocation obtained in these two runs (GB*sec) was similar: 76,730 using the simulator vs. 77,056 in the real cloud.

[13] http://hadoop.apache.org/docs/r1.2.1/gridmix.html

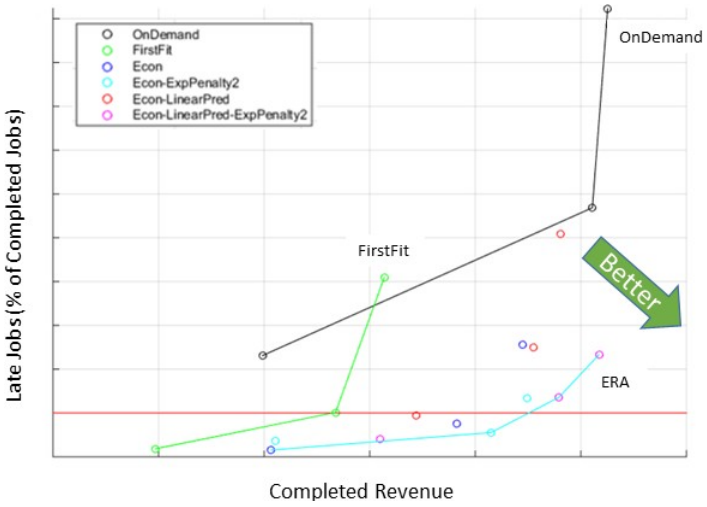

Figure 3: ERA over Azure Batch – simulation results (axis scales removed). ERA's econ al- gorithm dominates on-demand and first-fit algorithms in terms of the two desired measures of revenue and percentage of late jobs.

Testing Azure Batch The next set of simulations shows the advantage of using ERA over existing algorithms when applied on a cloud scale. In a typical cloud environment, we cannot expect one instance of ERA to have complete control of millions of cores. Thus, our goal here is to evaluate whether ERA will work with a subset of cores in a region, even while the underlying resource availability is constantly changing.

The simulations were of a datacenter consisting of 150K cores. ERA was given access to 20% of the resources and the remaining 80% were allocated to non-ERA requests, which were modeled using the standard Azure jobs. This means that resources were constantly being allocated/freed in the underlying region and ERA had

to account for this. The 20% of the resources under ERA's management came from the pre-emptible resources, but the design does not restrict its use to pre-emptible resources alone. ERA itself was run as a layer on top of the Azure Batch simulator, which simulates batch workloads on top of the Azure simulator of Microsoft.

ERA's Basic-Econ scheduling algorithm was experimented relative to two other algorithms: (1) the on-demand algorithm, which accepts jobs if there are enough available resources to start and run them (availability is checked only for the immediate time, ig- noring the duration that the resources are requested). It schedules accepted jobs to run immediately and charges a fixed price; (2) the greedy ("FirstFit") algorithm (described above), which charges the fixed, discounted, price of 65% of the non-pre-emptible re- sources price.

A common practice in the industry is to bound the maximal discount over non-pre- emptible machines. Accordingly, in our experiments ERA's Basic-Econ algorithm wasrestricted so that the price would be no higher than the non-pre-emptible jobs and would give no more than 35% discount. Several variants of the econ algorithm were explored:

(1) using either a linear predictor that is based on prior knowledge of the job distributions, or a predictor that uses past observations; (2) with or without an exponential penalty for later scheduling. Each of the variants was tested at a different capacity of the algorithm's use, so that the higher the capacity the fewer the resources that remained as spares for re-running failed jobs.

All jobs in the simulation workloads requested a time-window that started at their request-time (i.e., jobs did not reserve in advance). As ERA was getting 20% of the resources, we wanted to evaluate two measure metrics: (1) late-job percentage: this is the percentage of jobs that finished later than their deadlines; (2) accepted revenue: as we can charge only for jobs that are accepted, the better the algorithm, the more jobs we can accept. Figure3shows that ERA's econ algorithm dominates the other algorithms in terms of these two desired measures.

5. Grand Challenges

Clearly, the main challenge is to get the ERA system integrated in a real cloud system, and interface with real paying costumers. Short of this grand challenge, there are many research challenges. In this section we describe several challenges of a practical and theoretical nature related to the ERA (Economic Resource Allocation) project.

5.1. Job Scheduling

There is a vast literature on job scheduling both in the stochastic and adversarial models. The most obvious related model is job scheduling with laxity, which is the difference between the arrival time and the latest time in which the job can be scheduled and still meet the deadline. The current issues that are raised by our framework give rise to new challenges in both domains. In our setting it is very reasonable to assume that any job requires only a small fraction of the total resources, and that the laxity is fairly large com- pared to the job size. An interesting realistic challenge is to have a job give a tradeoff between time (to run) and resources (number of machines), which depends on the degree of parallelism of the job. Another interesting challenge is to exhibit a model that inter- polates between the stochastic model, which gives a complete model of the job arrival process, and the adversarial model, which does not make any assumptions. It would be nice to have a model that would require only a few parameters and be able to capture many arrival sequences. Finally, jobs of a reoccurring nature would be very interesting to study both in the stochastic and adversarial models.

5.2. Pricing

In our model we assume that the user has both a clear deadline in mind and an explicit bound on the length of the job. It would be interesting to give a more flexible guarantee, which would help the user to set his preferences in a less conservative way. For example, one could allow the job to run after it exhausts its resources at a certain cost and at a slightly lower priority for a certain additional amount of time. Another similar guarantee is that the user would have his “preferred deadline” and his “latest

deadline" with a guar- antee that most jobs finish at the preferred deadline. All this is aimed at a more flexible Quality of Service (QoS) guarantee by the system. Pricing such complex guarantees is a significant practical and theoretical challenge.

From the theoretical side, it would be nice to give theoretical guarantees to our sys- tem. First, to show that the users have an incentive to report their information truthfully, and not to try and game the system, or at least achieve this approximately. Second, to show that the system reaches a satisfiable steady state (e.g., showing an appropriate equi- librium notion and a related price of anarchy).

5.3. Learning

Our proposed framework requires a significant component of learning. Much of the learning depends on the observed time series from the past that would be used to predict future requests. A clear challenge in our setting is to accommodate seasonality effects (daily, such as day versus night; weekly, such as work week versus weekend; annual, such as holidays). Such challenges are well known in the time-series literature. A more interesting effect is that we have a system where the available resources and the demand are constantly growing, and the challenge is to bundle the two forecasts or somewhat separate them. It seems that our prediction model would need a more refined prediction than only the expected value, but for many of our forecasting applications we need to get more detailed information.

An additional uncertainty is that our system might be unable to see certain requests since the user decides that they were unlikely to be accepted and therefore never submit- ted them. For example, if a more important job is already rejected due to a low value, less-important jobs might be not submitted, and thus the prediction of the demand is even more challenging, given this partial information.

Finally, learning should not be limited only to the forecast of demand, but should also forecast the accuracy of the requests. Since in the current system we require that the job will not exceed its maximum length, it is likely to be a conservative estimate, and learning what is the "actual" demand might free significant resources.

5.4. Robustness

For any practical system to run it needs a significant level of robustness. Robustness should take into account both planned and unexpected failures in the various resources. Modeling this might be done as part of the greater challenge of a QoS guarantee. We should study what kind of an extreme-case guarantee can we give.

References

[1]Apache Hadoop Project. http://hadoop.apache.org/.

[2]V. Abhishek, I. A. Kash, and P. Key. Fixed and market pricing for cloud services. *arXiv preprint arXiv:1201.5621*, 2012.

[3]O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing amazon ec2 spot instance pricing. *ACM Transactions on Economics and Computation*, 1(3):16, 2013.

4. Amazon. Amazon elastic mapreduce. At http://aws.amazon.com/ elasticmapreduce/.

[5]M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson,
A. Rabkin, I. Stoica, et al. A view of cloud computing. *CACM*, 53(4):50–58, 2010.

[6]Y. Azar, I. Kalp-Shaltiel, B. Lucier, I. Menache, J. S. Naor, and J. Yaniv. Truthful online scheduling with commitments. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, pages 715–732. ACM, 2015.

[7]P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 164–177. ACM, 2003.

[8]E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *OSDI*, pages 285–300, Broomfield, CO, Oct. 2014. USENIX Association.

[9]Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The case for evaluating mapreduce performance using workload suites. In *Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2011.

[10]C. Curino, D. E. Difallah, C. Douglas, S. Krishnan, R. Ramakrishnan, and S. Rao. Reservation-based scheduling: If you're late don't blame us! In *SoCC*, 2014.

[11]A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca. Jockey: guaranteed job latency in data parallel clusters. In *Proceedings of the ACM European Conference on Computer Systems*, EuroSys, 2012.

[12]A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI*, volume 11, pages 24–24, 2011.

[13]R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource pack- ing for cluster schedulers. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 455–466. ACM, 2014.

[14]A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM computer communication review*, 2008.

[15]B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and
I. Stoica. Mesos: a platform for fine-grained resource sharing in the data center, 2011.

[16]N. Jain, I. Menache, J. S. Naor, and J. Yaniv. A truthful mechanism for value-based scheduling in cloud computing. *Theory of Computing Systems*, 54(3):388–406, 2014.

[17]S. A. Jyothi, C. Curino, I. Menache, S. M. Narayanamurthym, A. Tumanov, and et. al. Morpheus: Towards automated slos for enterprise clusters. In *OSDI*, 2016.

[18]K. Karanasos, S. Rao, C. Curino, C. Douglas, K. Chaliparambil, G. M. Fumarola, S. Heddaya, R. Ramakrishnan, and S. Sakalanaga. Mercury: Hybrid centralized and distributed scheduling in large shared clusters. In *ATC*, 2015.

[19]C. Kilcioglu and J. M. Rao. Competition on price and quality in cloud computing. In *WWW*, 2016.

[20]I. Menache, A. Ozdaglar, and N. Shimkin. Socially optimal pricing of cloud computing resources. In *ICST Conference on Performance Evaluation Methodologies and Tools*, 2011.

[21]I. Menache, O. Shamir, and N. Jain. On-demand, spot, or both: Dynamic resource allocation for executing batch jobs in the cloud. In *11th International Conference on Autonomic Computing (ICAC 14)*, pages 177–187, 2014.

[22]P. Menage, P. Jackson, and C. Lameter. Cgroups. *Available on-line at:* `http://www. mjmwired.net/kernel/Documentation/cgroups.txt`, 2008.

[23]D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[24]K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. Sparrow: Scalable scheduling for sub- second parallel jobs. Technical Report UCB/EECS-2013-29, EECS Department, University of California, Berkeley, Apr 2013.

[25]J. Rasley, K. Karanasos, S. Kandula, R. Fonseca, M. Vojnovic, and S. Rao. Efficient queue management for cluster scheduling. In *Proceedings of the Eleventh European Conference on Computer Systems*, page 36. ACM, 2016.

[26]D. Sarkar. Introducing hdinsight. In *Pro Microsoft HDInsight*, pages 1–12. Springer, 2014. [27]B. Sharma, R. K. Thulasiram, P. Thulasiraman, S. K. Garg, and R. Buyya. Pricing cloud
compute commodities: a novel financial economic model. In *IEEE-CCGRID*, 2012.

[28]Y. Song, M. Zafer, and K.-W. Lee. Optimal bidding in spot instance market. In *INFOCOM, 2012 Proceedings IEEE*, pages 190–198. IEEE, 2012.[29]M. A. team. Azure batch: Cloud-scale job scheduling and compute management. In `https://azure.microsoft.com/en-us/services/batch/`, 2015.

[30]A. Tumanov, T. Zhu, J. W. Park, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger. Tetrisched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clus- ters. In *Eurosys*, 2016.

[31]V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *ACM - SoCC*, 2013.

[32]A. Velte and T. Velte. *Microsoft virtualization with Hyper-V*. McGraw-Hill, Inc., 2009.

[33]A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at google with borg. In *Eurosys*, 2015.

[34]C. A. Waldspurger. Memory resource management in vmware esx server. *SOSP*, 2002.

[35]M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In *Eurosys*, 2010.