从多个集群的失败中学习：一种跟踪驱动的方法，用于了解，预测和缓解工作终止

摘要—在大型计算平台中，作业易于中断和过早终止，从而限制了它们的可用性并导致群集资源的大量浪费。在本文中，我们分三步解决这个问题。首先，我们基于来自多个大型生产系统的日志数据进行了全面的研究，以确定跨不同集群的不成功工作的行为模式，并调查工作终止的可能根本原因。我们的结果揭示了一些有趣的属性，这些属性将不成功的工作与其他工作区分开来，尤其是资源消耗模式和作业配置设置。其次，我们设计了一个基于机器学习的框架来预测工作和任务的终止。我们显示，可以相对较早地以较高的准确性和召回率预测工作失败，并且还可以识别具有强大的工作失败预测能力的属性。最后，我们在一个具体的用例中演示了如何使用我们建议的有效任务克隆策略将预测框架用于减轻执行不成功的影响。

介绍

在大数据时代，科学家，企业和政府都严重依赖有效地处理不断增长的数据量的系统和框架。尽管大型系统在处理数据量方面的能力以惊人的速度提高了，但可靠性和可靠性却并没有保持一致。实际上，对于不同类型的大型系统，可靠性的关注正在上升：无论是在传统高性能计算（HPC）平台中，应用程序预期在下一代计算机中每隔几分钟就会中止[16]，还是在云计算中在复杂软件和异构硬件之上运行各种工作负载的环境，增加了作业出现错误和错误的可能性[5]，[8]，[10]，[13]。

最近的几项研究分析了Google公开提供的针对其多用途集群之一的大型工作负载跟踪，[17]花费了大量的集群时间来执行无法成功终止的工作（例如，失败或被杀死）[5]， [8]，[10]，[13]强调需要深入了解工作在大型系统中终止的方式和原因。

尽管这些研究提供了有关Google集群执行不成功的特征的一些见解，但还没有后续工作来调查报告的发现对其他计算集群或其他大型系统的普遍性。此外，还没有任何工作能够证明如何将这些研究的结果用于减轻不成功的工作执行或以其他方式改善集群的工作。

在这项工作中，我们的首要目标是使用在不同系统上收集的工作负载跟踪信息，全面分析工作终止的特征和根本原因。我们利用来自三个不同来源的跟踪：Google的多用途群集[17]，卡内基梅隆大学（CMU）的Hadoop群集[2]和洛斯阿拉莫斯国家实验室（LANL）的HPC群集。使用这些痕迹，我们调查了大型作业如何以及为什么在并行集群中无法成功终止，并且基于我们的观察，我们提出并评估了作业失败预测和缓解的框架。

本文的主要贡献如下：

•我们提供了将未成功执行工作与成功执行区别开来的因素的详细描述（第三节），并探讨了可能解释执行失败的根本原因的不同假设（第四节）。我们的工作将先前的工作扩展到包括两个新的大型系统的数据集，并涵盖了先前工作中未探讨的因素。对于一些先前研究的因素，我们从先前的工作中得出了不同的结论。

•我们使用机器学习技术来预测工作和任务级别的执行失败。我们表明，可以在工作生命周期的早期相对准确地预测出执行失败的情况，并具有很高的准确性和召回率（第五节）。

•我们提出并评估了一个实际的用例，以说明如何使用我们的发现和预测变量来通过任务克隆减轻工作失败，并讨论了许多其他潜在用例（第6节）。

工作轨迹说明

我们的研究基于在三个不同组织中收集到的以下工作跟踪：

a）Google：该跟踪是公开可用的[17]，捕获了一个月内提交给Google的12,000个节点多功能计算集群的所有34.9万个工作。作业可以包含一个或多个任务，并且对于每个任务以及整个作业，将记录退出状态：如果作业和任务成功完成，则标记为“完成”；如果用户中止或由于对不再满足的另一项工作的依赖而将其“杀死”；如果由于较高优先级任务到达同一台计算机而导致任务调度，则“退出”；和“失败”，通常是软件崩溃导致的。在最终退出之前，任务可以进行多次尝试，每次尝试也都具有一个状态。有关跟踪的详细信息，请参见[9]。

b）CMU OpenCloud：OpenCloud [2]是卡内基梅隆大学的一个64节点Hadoop集群，研究人员在计算生物学，图像处理和机器学习等领域使用了该集群。跟踪记录是在30个月的时间内收集的，涵盖了7.8万个作业，并包含有关作业配置和资源使用情况以及群集中每个作业的退出状态的信息：“成功”（如果成功完成）；如果用户手动中止，则为“已杀死”；并在作业崩溃时“失败”。有关跟踪的详细信息，请参见[2]。

c）LANL HPC集群：LANL为256节点HPC系统（其网页上ID为20的系统[3]）公开提供了为期三年的作业跟踪记录，科学家通常使用它来运行CPU和内存密集型的科学模拟。这些跟踪包含有关作业启动时间和退出时间的数据，以及每个作业的最终状态：所有应用程序过程成功完成时“完成”；如果用户取消了工作，则“放弃”；如果应用程序被信号终止，则被“杀死”；如果工作因维护或问题而被管理用户杀死，则为“ syskill”；如果运行此作业的一个或多个节点崩溃，则为“失败”。有关跟踪的详细信息，请参见[3]，[15]。

表I总结了每个群集中的跟踪特征以及作业退出状态的细分。在集群时间方面，我们发现执行失败会消耗大量计算周期（每个跟踪中超过50％的集群时间）。

在大型集群中，什么特征可以使未成功的工作成为特征？

我们首先研究跨不同集群的不成功工作的特征。我们的目标是利用上述跟踪信息来了解失败或取消的作业执行如何表现出将其与成功执行区分开的模式。我们检查了各种作业属性，包括作业持续时间，并行度和群集资源利用率。

a）作业持续时间：我们将作业持续时间定义为该作业在退出之前在集群中花费的总任务分钟数。对于作业不包含任务的LANL群集，我们使用节点分钟数来表示作业持续时间。在表II中，我们比较了每个群集中失败的作业，被杀死的作业和成功完成的作业之间的作业持续时间分布。我们观察到，在所有三个集群中，成功的作业始终在同一集群中安排的所有作业中持续时间最短，而被杀死的作业在终止之前始终花费最长的时间。这种观察是直观的，因为例如，当用户花费比平常更长的时间来完成工作或停止响应健康监控信号时，用户将中止工作。最后，我们发现，对于LANL HPC群集，失败的作业报告的持续时间与已终止的作业相当，但是在Google和OpenCloud中，失败的作业比已终止的作业短。

b）并行度：在Google集群中，35％的批处理作业是多任务作业，其中89.3％最终被杀死，而90％以上的单任务作业已完成。对于单任务和多任务作业，失败作业的比例约为1.5％。在OpenCloud中，任务要么是“映射器”，要么是“减速器”：超过80％的工作中至少有2个映射器，近60％的工作有1或0个减速器。我们使用映射器的数量来表示作业的并行度。与Google不同，我们发现OpenCloud中单任务和多任务作业的工作状态细分几乎相同，并且反映了整体细分（请参阅表I）。在LANL群集中，79％的作业在单个节点上运行。我们发现单节点和多节点作业的成功率分别为70％和50％。单节点作业中被杀死或被系统杀死的作业比例（22％）在并行作业中几乎翻了一番（45％）。

由于并行作业在集群中花费了更多时间，因此它们自然更有可能遇到问题。为了解决此问题，我们通过跟踪每个类别中的作业所消耗的任务周（或节点周）的总数来归一化在跟踪中观察到的作业中断数。图1显示了我们数据集中所有作业的结果。

出乎意料的是，我们发现当按系统时间进行标准化时，与单个作业相比，并行作业每单位时间的作业中断率较低（LANL群集中唯一被杀死的作业除外）。

c）群集资源的使用：现在，我们研究完成的作业与不成功的作业如何使用群集资源。我们首先检查提交时作业最初请求的资源量。在Google中，可以提供有关任务任务所请求的CPU，内存和磁盘空间量的数据。在OpenCloud群集中，作业的配置文件中提供了作业中每个映射器和每个reducer所需的虚拟内存（VM）量数据。在LANL群集中，每个作业提交都提供了有关请求的处理器数量的数据。

图2分别绘制了完成任务，失败任务或被杀死任务的任务平均请求资源的分配情况（对于LANL任务，仅按请求数目的处理器）。

我们观察到的第一件事是，资源请求在我们数据集中所有集群中的成功和不成功作业之间明显不同。在Google中，失败的作业需要比其余作业更多的内存和磁盘空间容量。在OpenCloud集群中，完成或失败的作业中有99％以上为映射器和化简器请求的VM数量保留为默认值（1024 MB [1]）。被杀死的作业是一个例外，因为它们配置的VM请求与默认设置有很大不同。在LANL，失败或被杀死的作业报告的群集中对处理器的请求明显高于其他作业。我们的观察结果表明，对作业请求的群集资源数量的了解可以潜在地用于预测作业的最终状态。 （我们在第V节中研究了执行失败的预测。）

在检查了资源请求之后，我们现在将注意力转移到这些作业在运行时实际上如何消耗群集的资源。有关资源利用率的数据仅可从Google和OpenCloud获得。在Google中，每项任务都可以使用CPU，内存和磁盘使用情况的数据（每隔5分钟汇总一次）。

OpenCloud跟踪仅包含有关I / O任务使用情况的数据，这些数据是从Hadoop的分布式文件系统（HDFS）和节点本地文件系统计数器收集的。我们检查了四个I / O计数器：作业开始时映射器读取的字节数（HDFS\_BYTES\_READ）；产生最终输出时，减速器写入的字节数（HDFS\_BYTES\_WRITTEN）；在reducer节点上从本地文件系统读取的字节数（FILE\_BYTES\_READ）；以及在生成映射输出时由映射器写入本地磁盘的字节数，以及在混洗阶段（FILE\_BYTES\_WRITTEN）上由reducer溢出的字节数。

图3研究了Google和OpenCloud集群中作业的集群资源平均利用率分布。我们观察到成功和不成功的工作之间存在不同的资源使用行为。在Google，被杀死的作业占用了更多的CPU和内存，而失败的作业报告了更高的I / O利用率。值得注意的是，此处的I / O消耗是指节点本地磁盘的使用情况。 Google的踪迹不包含有关GFS（Google文件系统）使用情况的任何信息。

在OpenCloud中，我们还观察到已完成和中断的作业之间存在不同的I / O使用行为。最值得注意的是，我们发现失败的作业报告了对工作节点中本地磁盘的I / O写入率最高。这与我们对Google集群的观察结果一致，在该集群中，失败的作业还报告了所有作业中的本地磁盘使用率最高。

不成功的执行有哪些可能的根本原因？

现在，我们使用这些痕迹来调查不同的假设，这些假设可以揭示大规模集群中终止工作背后的根本原因。

A.违反资源限制

一种可能的假设是作业由于资源不足而无法成功执行。例如，在Google，资源管理器可以终止使用比开始时请求更多的内存的任务，并且可以限制使用CPU时间超过请求数量的任务的任务[9]。我们有兴趣了解这种资源冲突问题在现场发生的频率，从而导致工作或任务终止。我们使用来自Google的跟踪信息来探索这个问题，因为它是唯一可为集群中每个任务的CPU /内存限制和CPU /内存使用情况提供信息的数据集。

我们为跟踪中的每个任务计算任务在其生存期内平均消耗的CPU /内存量与任务在提交时请求的CPU /内存量之间的比率。图4-（a）绘制了该比例在Google任务中的分布情况，并按任务退出状态进行了细分。另外，我们考虑了任务执行期间报告的最大CPU /内存量与任务请求的CPU /内存之间的比率，如图4-（b）所示。图中水平虚线上方的数据点（指比例= 1）对应于超出其资源限制的任务。

我们发现，在一般情况下，任务很少超过其内存限制，但是25％的失败任务在其生命周期的某个时刻确实超过了其内存限制（请参见图4-（b）中的右图）。另一方面，对于此群集中的任务而言，消耗比最初请求更多的CPU周期也不成问题。实际上，成功完成的任务比其他任务更有可能超出其CPU请求。

为了更好地了解任务在运行时的内存使用率何时飙升，例如由于内存泄漏，我们针对Google的所有批处理任务计算了任务生命周期后半段的平均内存使用量与其上半年平均内存使用量之间的比率。我们发现，随着时间的流逝，失败的任务中有22.87％的平均内存消耗增加了一倍以上（比率大于2）。同时，被杀死的任务和完成的任务的13.5％和9.68％的报告比率大于2。

观察：对于我们数据集中的Google集群，内存管理问题与任务失败相关。尽管失败的任务比其他任务需要更多的内存（请参阅第III部分），但它们仍然更有可能超过其内存限制，并且随着时间的推移会经历内存使用的急剧增长。

B.作业容错配置

我们要问的下一个问题是，将作业配置为在运行时处理问题（例如任务缓慢或失败）的方式是否与作业终止相关。

在Google和OpenCloud集群中，底层框架通过数据复制和任务重新执行（即任务在失败时重试）来提供可靠性和容错能力。在LANL群集中，作业表示紧密耦合的HPC应用程序，其中主要通过应用程序级检查点来实现容错。 LANL跟踪不为这些作业的检查点参数提供任何数据。 Google也不提供有关作业容错配置的信息，但是可以使用任务事件日志来分析运行时任务重新执行对作业可靠性的影响。但是，CMU OpenCloud跟踪包含有关作业和任务配置的详细信息（从Hadoop的作业配置XML文件中收集）。接下来，我们将研究用于配置Hadoop的“推测执行”功能，处理不良数据记录和管理任务重试的容错旋钮。

a）OpenCloud中的推测执行（SE）：Hadoop中的SE是一种处理并行作业中的映射器或化简器任务最终出现在慢速节点上的情况，因此会降低整个作业的速度。然后，框架将在多台计算机上计划慢速任务的冗余克隆，并最终从完成的第一个克隆中收集输出。用户可以通过取消设置以下SE标志来禁用其工作的SE功能：map.tasks.speculative.execution和reduce.tasks.speculative.execution。通过分析OpenCloud配置文件，我们发现99％的计划作业使映射器和化简器均启用了此功能。当按退出状态划分作业时，我们发现虽然分别只有0.02％和0.5％的已完成作业和已杀死的作业禁用了映射器和/或化简器的SE，但2％的失败作业中有很大一部分禁用了此功能（在对映射器和缩减器均禁用SE的作业与仅对映射器禁用SE的作业之间几乎均分。

b）OpenCloud中的“跳过”模式：“跳过”模式是Hadoop识别可能导致任务失败的错误输入数据的一种方法。在任务的尝试次数达到一定限制（通过参数mapred.skip.attempts.to.start.skipping配置）后，框架启用此模式，此后任务必须向框架报告它将处理的数据记录下一个。如果任务最终失败，框架将知道哪些记录可能是有问题的记录，以便在以后的尝试中跳过它们。在OpenCloud集群中，我们发现96％的作业中启用了跳过模式的任务尝试限制保留为默认值2次尝试。在考虑作业的退出状态时，我们发现已完成作业，杀死的作业和失败的作业分别有96.5％，97.6％和95.5％具有此参数的默认配置，第二受欢迎的值是16工作状态组。

c）OpenCloud中任务重试的限制：现在，我们考虑使用框架mapred.map.max.attempts和mapred.reduce.max配置的映射程序任务或化简任务在框架终止之前可以进行的最大尝试次数。尝试。在OpenCloud集群中，所有作业的4％修改了映射器任务的默认限制（4次尝试），而只有1％的作业修改了reducer任务的重试限制。图5显示了OpenCloud作业中配置的任务重试限制的详细细分。我们发现，对于映射器，此限制的值高达100，对于缩减器，该值高达130。条形图显示，对于映射器和化简器，被杀死的作业占工作重试的最大比例，其重试限制均超过默认设置，其次是失败的作业。

观察：对于具有配置数据的CMU Hadoop集群，不成功的作业更有可能具有与默认框架设置不同的容错配置；例如推测性执行的禁用和允许的任务重试限制的增加。

C.导致作业失败的任务失败

在本节中，我们研究任务失败在作业失败中扮演的角色。

Google和CMU分别有超过98％和95％的失败工作具有至少一项失败任务。相比之下，分别在Google和CMU完成的工作中只有0.25％和20％具有一项或多项失败的任务。这促使我们回顾关键机制对任务级容错的有效性，该机制包括重试失败的任务。

我们首先问一个问题：在一次或多次失败的尝试之后，任务重试成功的可能性如何。图6绘制了针对Google（顶部）和OpenCloud（底部）的任务下一次尝试成功的概率与过去失败尝试次数的函数。

值得注意的是，在Google和OpenCloud中，一次成功完成任务而没有过去失败（图中未显示）的概率分别为98％和97％。我们从图6的最左侧图表观察到，在第一次重试任务（即尝试失败一次）后，谷歌和OpenCoud的概率分别急剧下降至42％和28％。在第二次重试中，任务的成功机会在Google中仅下降到2％，在OpenCloud中下降到11％。在两个集群中，经过两次或三次以上的尝试后成功的可能性变得微不足道。

有趣的是，当仅考虑单任务作业（最右边的图）时，我们发现成功执行第一个任务重试的可能性比并行作业（中间图）要低得多：在Google和OpenCloud中分别为13％和15％。请注意，单任务成功而没有失败的概率（图中未显示）在Google中为97％，在CMU中为90％，因此，这些较低的重试机会无法用较低的初始成功机会来解释。对于单任务作业中此观察结果的一种可能解释是缺少竞争条件，该条件可能会影响并行任务的完成，因此使单任务作业更有确定性地失败。

尽管上面的观察表明，重复执行两次以上的任务几乎是徒劳的，但我们发现许多用户过于乐观，从而浪费了大量的群集资源：我们在Google和OpenCloud中研究了失败任务中#task重试的分布，发现70-90％的失败任务尝试重新执行一次以上，而15-30％的失败任务尝试重新执行两次以上。一些失败的任务在终端出现故障之前尝试重新执行多达70–100次。

观察：对于我们的数据集中以任务重新执行为恢复机制的集群，我们发现尝试重新执行一次或多次以上的任务是徒劳的。但是，我们发现，在最终失败的任务中，重试次数较多，并且任务失败与整个作业失败密切相关。

D.机器中断

造成作业中断的另一个可能的根本原因是托管作业的物理机（节点）发生故障。在我们的数据集中，有关节点中断的信息可用于Google和LANL集群。在Google上，可以获得在数据收集期间发生的群集中所有计算机删除的记录。机器的“拆卸”可能是由于计划内的维护或计划外的故障。根据Google [17]的说法，每当移除一台计算机时，在该计算机上运行的任何任务都会记录一个任务“退出”事件，并且框架将尝试将该任务重新安排在另一台计算机上。在LANL，还可以提供在跟踪收集期间发生的所有节点中断的记录。根据LANL网站[3]，当节点发生故障时，在该节点上运行的所有作业都应以“失败”状态退出群集。

通过将机器事件的日志与作业事件的跟踪相关联，我们检查了节点中断对Google和LANL集群中作业可靠性的影响。我们对量化可归因于机器故障的工作终止比例特别感兴趣。因此，我们使用记录的时间戳来识别Google的所有任务以及LANL的所有作业，这些作业的执行由于机器中断事件而中断。

表III列出了与机器拆卸相关的作业/任务事件的类型细分（左栏）；我们还显示了每种事件类型，此类型事件总数中有多少比例与机器故障相关（右列）。例如，在Google集群中，我们发现91％被Google移除计算机中断的任务事件是任务“逐出”，而这些与机器相关的逐出占跟踪中出现的所有任务逐出的16％ （其余的驱逐很可能是由于更高优先级的任务抢占而来）。

在Google群集中，随着计算机移除而出现的第二高事件类型是任务终止（8.4％）。一种假设是，这些故障可能与突然的机器故障有关，在机器突然关闭之前没有向任务发送适当的终止信号。有趣的是，当我们转向LANL集群时，发现与计算机关闭相关的作业事件中有90％以上是“系统被杀死”，“被杀死”或“中止”。任务和作业“失败”分别仅占Google和LANL所有与机器移除有关的事件的0.3％和2％。在Google集群中，这些任务失败仅占跟踪中出现的所有任务尝试失败的一小部分（0.004％）。在LANL，与计算机故障相关的作业失败占集群所有作业失败的5.4％。

观察：对于我们的数据集中有机器停机日志的集群，我们发现与机器移除相关的大多数任务/作业事件是逐出或杀死。在Google集群中，很少将机器删除视为任务失败背后的根本原因，并且可以解释LANL集群中约5％的工作失败。

预测不成功的执行

在研究了不成功的作业的属性并调查了终止工作的可能根源之后，我们现在提出一个问题，即是否可以预测运行时作业的不成功终止。我们主要关注Google跟踪以设计和评估我们的预测框架，因为这些跟踪包含有关作业和任务执行的详细信息以及CPU，内存和I / O使用情况的高分辨率日志。

A.预测框架的设计

我们的目标是设计一个能够预测集群中正在运行的作业或任务未成功终止的框架。表IV总结了我们用作预测变量的输入变量。 “配置”下的字段是指变量的值，这些变量的值在作业配置时（即在执行之前）为已知。 “计数器”和“用法”下的字段仅在运行时才知道。

预测技术：我们使用基于决策树的机器学习技术随机森林[4]（RF）对工作和任务进行预测。我们将RF算法配置为在每次迭代中使用50棵树，并对数据进行拆分，以使跟踪中Google的工作中有70％用于培训，而30％用于测试。

预测旋钮：我们使用默认的分类（决策）阈值0.5来预测类成员身份，但是在我们的某些实验中，我们将阈值从0.1更改为最大0.9。对于我们的训练集，我们对数据点的正负类别之间的各种比率进行实验，以校正原始数据中的偏差（因为成功事件与失败事件之间的比率不平衡），这种技术通常称为“数据超采样”。

指标：我们用来量化预测质量的指标是精度和召回率，这是两个标准的机器学习指标。精度是指真实肯定的数量除以正确肯定和错误肯定的数量。回忆是真实肯定的数量除以真实肯定和错误否定的数量。例如，当预测某项工作是否会失败时，精度指的是在所有预测为失败的工作中实际确实失败的工作所占的比例，而召回率则是指预测出的所有工作中会失败的工作所占的比例所有确实失败的工作。

请注意，在一些先前的工作[6]，[11]，[12]中使用的另一个通用度量标准，准确性，对于我们尝试预测的事件类型而言，并不是一个有趣的度量标准。准确度是真实的正数加上真实的负数除以整个人口（= 1-误分类率）。对于罕见事件，始终做出负面预测的预测器将具有较高的准确性，但在实践中不是很有用。

预测场景：我们从最简单的场景开始尝试不同的场景：我们能否仅使用作业提交时可用的数据来预测作业的不成功终止？我们将这种情况称为“仅配置”。我们探索的第二种情况假设了解该作业的早期资源使用指标，尤其是在执行的前5分钟内（除了配置参数之外）该作业任务的CPU，内存和磁盘消耗的平均和标准偏差。 。对于运行时间少于5分钟的作业，我们注意到这些短暂任务的一部分在跟踪中记录了有效的使用情况日志（例如，在执行的前2或3分钟内）。我们将这些短暂任务与使用情况数据包括在内实验。我们将第二种情况称为“配置+ 5分钟使用情况”。

在第三个预测方案中，我们添加了一个运行时标志，一旦在作业中的所有任务中检测到单个任务失败，就将其设置为TRUE。我们将此设置称为“配置+ TaskFailFlag”。最后，第四种情况假设您已了解上面讨论的所有参数：“ Config + 5min用法+ TaskFailFlag”。

B.作业被杀死预测

我们首先探讨在Google中是否可以准确预测职位“技能”。回想一下，在Google集群中执行kill操作可能表明用户手动中止操作或由于不满意的依赖关系而终止了操作。一旦一项工作被杀死，其所有任务也将立即被杀死。图7显示了在我们不同预测变量下获得的精度和召回率；左图显示的是将分类阈值从[0.1-0.9]更改时的结果，右图显示的是在固定阈值0.5下更改训练集中的采样率时的结果。

我们发现，即使在仅使用作业配置参数的最简单情况下，也可以非常准确地预测作业终止。在仔细研究决策树的输出时，我们发现作业请求的CPU，内存和磁盘的数量以及作业的并行度被标记为作业终止的最重要预测指标。我们还观察到，对于工作死亡的预测质量比对训练抽样率对分类阈值更为敏感。接下来，我们探讨是否可以实现相同水平的高质量预测来预测工作失败。

C.工作失败预测

图8显示了运行我们的工作失败预测器时使用的精度与召回率，使用与上述工作失败预测相同的配置。我们观察到了在不同设置下获得的预测质量的高得多的可变性，以及更广泛的精确调用折衷范围。单独使用作业配置数据将不再产生与杀死作业相同的出色结果。但是，有趣的是，仅查看工作的配置方式，我们就可以以高达79.5％的精度预测某个工作将失败，从而召回所有失败工作的50％以上。仔细研究发现，作业请求的磁盘空间量是决策树使用的重要预测指标。

我们发现，考虑工作的前五分钟资源利用率，可以在某种程度上改善预测效果，但是，当包含TaskFailFlag时，可以实现最显着的改进。使用此标志，我们可以以至少95％的精度成功调用多达94％的所有失败作业。

为了更好地了解此预测变量在实践中可带来多少收益，我们研究了在工作生命周期中第一次任务失败发生的时间。图9绘制了第一个任务失败发生后并行作业中的剩余时间（左）和并行作业寿命中的剩余部分（右）的分布。我们发现，失败工作的平均离职时间约为60分钟，而死亡工作的平均离职时间则为一整天。就剩余的工作部分而言（图9-（右）），我们发现所有工作状态组的中位数约为60％。也就是说，在第一个任务失败事件发生之前，有失败任务的作业中有一半的运行时间约占其总执行时间的40％或更少。该结果表明，可以及早准确地预测工作失败。

鉴于单个任务失败是工作失败的有力预测因素，接下来要问的自然问题是我们是否可以预测工作中任务的失败。

D.任务失败预测

回想一下，如果第一次执行任务的尝试失败而失败，则可能会重试该任务几次。我们首先在第V-D1部分中预测任务的最终状态，即任务退出时（可能在多次任务重试之后）的状态，然后在第V-D2部分中预测预测中间重试尝试的状态。

1）预测最终任务状态：除了上述方案外，我们还添加了一个AttemptFailFlag预测器，如果过去至少一次对该任务的尝试失败，则将其设置为true。图10（a）显示了任务级别的结果。我们观察到，与作业相比，仅使用配置数据会产生较不精确的预测，并且与用于作业级预测时相比，了解前几分钟的任务资源使用情况对预测的改善更大。原因可能是作业的视图比任务的分辨率低，因为它考虑了作业中所有共同任务的平均消耗。最后，我们发现使用AttemptFailFlag可以最大程度地提高预测效果，特别是对于我们的召回率。这种观察使我们想到了下一个问题：我们可以在任务尝试级别上预测失败吗？

2）预测任务尝试失败：图10（b）显示，与任务级和作业级失败相比，可以更精确地预测单个任务尝试的失败。决策树将请求的内存量，请求的磁盘空间和并行度标记为最重要的预测指标。

减轻执行失败的影响

在本节中，我们探索如何将我们的工作结果和预测框架用于减轻不成功的工作执行的影响。我们从第VI-A节中提出了一系列不同的建议开始，然后在第VI-B节中提出了一个特定的用例，在该用例中，我们将其中一个建议发展为具体算法，并评估了其在跟踪驱动的仿真中的有效性。

A.政策建议

先前关于预测任务失败和死亡的工作完全集中在使用这些预测来主动终止可能不成功的工作。我们认为，此方法的实用性在实践中至少有两个原因，值得怀疑：首先，如果系统杀死了他们的工作，则用户可能会感到不满意，唯一的原因是系统认为工作不太可能成功。它还可能做出错误的假设，即最终失败的作业对用户没有任何价值：作业可能会在其整个运行期间连续产生输出，并且这些中间结果可能对用户有用。其次，主动取消工作使用户很难确定该工作出了什么问题。让作业运行直到它本身失败，可能会生成错误消息或调试信息，用户可以使用它们来排除故障并修复其作业。

相反，我们提出了一些替代建议，我们认为它们更实用，更有效：

•限制重试次数：我们的第一个建议很简单：鉴于重试失败任务的成功率随次数迅速下降为零对于尝试次数（请参见图6），我们建议对失败后重试任务的次数设置上限R，以限制浪费的资源量。

•启用其他监视：通常，作业可以生成许多其他监视和调试信息，但是出于效率原因，通常在生产运行期间禁用此类信息的收集。我们的预测框架的一种潜在用途是，一旦预测到执行不成功，就可以动态地启用其他信息的收集，以便用户以后使用该信息来解决问题。

•增加检查点的频率：许多程序依赖于定期检查点，这些检查点可用于在发生故障时进行恢复，或为已终止/已终止的作业提供中间结果。一旦预测执行失败，我们的预测框架可用于打开检查点或增加检查点频率。

•调整计划优先级或分配的资源：鉴于作业的计划优先级[13]及其分配的资源都与其退出状态相关联，因此可以考虑根据退出状态预测来调整这些属性。此类策略的具体细节将取决于系统内部的详细信息，包括这些作业是什么以及使它们可能失败的原因。可能有人争辩说，应该减少不太可能成功的工作的优先级和资源，以减少它们对其他（更有希望的）并发工作和整体资源使用的负面影响。或者，可以辩称，对于那些被认为很重要的预计失败的工作，应增加其优先级和资源以提高其成功机会。

•计划冗余任务（推测执行）：由于没有更多的系统级信息很难提出降低故障率的建议，因此我们将重点放在最大限度地减少失败任务造成的损害的策略上。失败的任务的一个有害副作用是，它们导致大量的作业减速。因此，我们建议使用类似于用于处理散乱者的推测执行的策略：一旦预测任务可能以失败告终，我们将主动并行启动任务的克隆，而不是等到任务失败后再开始然后开始重试。虽然这不会增加任务成功的机会，但这意味着可以更快地达到最终执行状态。我们将详细研究这种策略的细节，包括在下一节中对它的好处进行跟踪驱动的评估。

B.用例：主动克隆

为了最大程度地减少因失败的任务引起的运行速度[13]，我们的目标是确保失败的任务能够尽快完成。我们建议使用第五节中的预测框架为可能失败的任务抢先开始早期重试尝试（即，重试尝试与第一次任务尝试重叠执行）。例如，对于在两次失败尝试后均以失败告终的任务，我们的目标是与第一次任务尝试并行地抢先启动该任务的克隆，而不是等到第一次尝试失败之后再依次开始第二次尝试。如果克隆足够早地启动，则可以预期会大大缩短完成克隆的时间。我们略微修改了预测框架，以使用滑动窗口在线监视任务资源的使用情况，而不是仅依赖最初的几分钟。

我们的克隆算法在算法1中进行了描述。输入参数包括要创建的重试总数的上限R和C（预测任务失败时允许创建的额外克隆数）。在本节的其余部分中，我们首先评估该方法在减少任务完成时间方面的有效性，然后评估其产生的开销。

1）评估任务加速：我们使用基于Google跟踪的模拟来评估使用主动克隆策略可以加速多少任务完成时间。在对随机选择的25,000个任务的训练集上的预测变量进行训练后，我们对另一个随机选择的25,000个任务集进行算法1的评估。我们通过完全按照跟踪中给定的方式重播任务来执行此操作，只是我们根据预测变量修改了重试的开始时间（即，一旦预测到失败，便开始重试）。我们假设单个任务尝试的执行时间和退出状态与原始跟踪中的相同，即不受更改其开始时间的影响。尽管实际上执行时间可能因一次运行而不同，但可以合理地假设期望值与平均时间相同。请注意，可以从我们的政策中受益的任务包括经历至少一次失败的任务尝试的所有任务，而与任务是最终成功（一次或多次失败尝试）还是永久失败（多次失败尝试）无关。

图11（a）和（b）显示了算法1下完成时间的累积分布函数（CDF），是轨迹中原始完成时间的一部分。在此实验中，我们将R设置为4（即最多4次重试），将C设置为1（即，在预测到故障时仅启动1个额外的克隆），并使用3个不同的阈值配置故障预测算法，从而改变主动性预测因子是。图11-（a）显示了最终退出状态为失败的作业的结果，图11-（b）显示了成功执行（但经过一次或多次失败的尝试）的作业的结果。我们观察到，即使对于阈值为0.9的保守预测变量，中位失败工作的完成速度也要快20％。更积极的预测不会显着改善结果。图11-（b）显示了最终成功的作业的结果，但是仅在一次或多次失败的尝试之后。在这种情况下，完成时间的减少较少，但仍然很重要。

我们还对克隆C的不同值进行了实验，并在图11-（c）中显示了结果。我们发现，在做出故障预测时，积极地启动多个克隆对完成时间几乎没有好处。但是，正如我们将在下一节中看到的那样，它可能会带来大量开销。

2）开销评估：请注意，对于一个完全准确的任务失败预测器，模拟系统中的工作量将与原始系统中的工作量相同，因为执行相同的工作的时间可能是不同的（重试发生）之前）。开销的唯一来源是预测器的误报，这会触发重试，尽管后来发现先前启动的尝试成功了。因此，开销将取决于预测器的攻击性（其阈值）和在每次失败预测之后开始的克隆总数（即参数C）。表V报告了算法1的不同配置带来的开销（系统上的总工作量的增加）。我们观察到，为了使开销保持合理，需要保守地选择阈值，范围在0.9或更高。

相关工作

最近，有几篇论文通过分析Google为其集群之一提供的工作负载跟踪，研究了大型系统中执行失败的性质[5]，[7]，[10]，[13]，[14]。 17]。此先前工作的一个主要限制是它仅关注单个集群，而没有调查观察到的不成功工作的特征是否可以推广到其他系统。我们的工作扩展了这些研究，扩展了数据集，除了Google跟踪之外，还包括来自不同组织的其他两个大型系统的日志，并探索了影响工作和任务可靠性的其他因素，而以前的工作并未考虑这些因素（例如，违反资源限制，作业容错配置，机器故障等）。我们发现先前报告的Google集群中作业失败的某些属性确实存在于其他系统中，例如相对较大的资源请求和成功执行任务重试的机会较低[13]。在其他集群中显示出较弱趋势的其他观察到的特性包括多任务作业中较高的作业中断率。另一组论文[6]，[11]，[12]主要使用神经网络[6]，[11]或线性/二次判别分析[12]来预测工作失败。所有这些论文提出了基于终止预期失败的作业的缓解故障的策略。我们提供了一种不同的预测技术，该技术比以前的工作在预测质量上有显着提高，并使用了更易于训练和在实践中使用的随机森林。我们的工作也是首次提出可使用预测器的多种实用方法，并详细介绍和评估利用分布式框架中可用功能的一个特定用例，这是我们的第一项工作。

结论

并行集群的可靠性在很大程度上取决于其有效处理故障的能力。本文使用来自不同生产系统的庞大的工作量跟踪数据集来研究并行集群中的工作终止。我们确定了可以区分在现场如何配置和执行不成功作业的模式。

我们发现，在多个群集中，不成功的作业会运行更长的时间，需要更多的群集资源，报告的I / O活动要比已完成的作业重，并且其参数配置通常会偏离框架的默认参数。我们的分析指出了一些可能导致作业失败的重要因素（例如，违反资源限制和内存泄漏），并确定了其他一些不太重要的因素（例如，机器故障）。

我们设计和评估工作和任务终止的有效预测器，并确定具有强大的工作或任务终止预测能力的参数。我们发现仅在启动时就知道的作业配置参数足以预测作业是否将被杀死，并且单个任务失败是作业失败的非常准确且早期的指标（尽管有内置方法恢复失败的任务，例如任务重试）。在在线资源使用的帮助下，我们还能够在生命周期的早期准确地预测单个任务尝试的退出状态。

最后，我们提出了多种方法来利用我们的结果来减轻集群中的失败，并提供对其中之一的详细评估：我们演示了如何有效地使用以失败预测为指导的任务克隆来限制执行失败的影响。