doi:10.3969/j.issn.1000-436x.2014.10.020

虚拟化云平台中实时任务容错调度算法研究

王吉，包卫东，朱晓敏
（国防科学技术大学 信息系统工程重点实验室，湖南 长沙 410073）

摘 要：为了在云平台下满足实时系统的高可靠性要求，提出了一种虚拟化云平台中的容错调度算法（FSVC, fault-tolerant scheduling algorithm in virtualized clouds），FSVC 通过主副版本方法来实现对物理主机的容错，采用副版本重叠技术与虚拟机迁移技术来提高算法的调度性能。为了达到容错的要求，分析了这 2 种技术应满足的约束。此外，FSVC 中包含了一种两阶段策略以进一步提高算法性能。大量仿真实验表明，在虚拟化云平台中，FSVC 能有效地提高系统可调度性与资源利用率。

关键词：虚拟化云平台；容错调度；主副版本方法；实时系统
中图分类号：TP393 **文献标识码**：A **文章编号**：1000-436X(2014)10-0171-10

Fault-tolerant scheduling algorithm for real-time tasks in virtualized cloud

WANG Ji, BAO Wei-dong, ZHU Xiao-min
(Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha 410073, China)

Abstract: In order to achieve the high reliability of real-time systems in virtualized clouds, a fault-tolerant scheduling algorithm for virtualized clouds named FSVC that used primary-backup approach to tolerate physical host failures was proposed. FSVC strived to enhance the performance by employing backup-backup overlapping technique and virtual Machine migration technique. The constraints of the two techniques were elaborately analyzed to realize fault tolerance. Besides, a two-phase policy was incorporated in FSVC to further improve the performance. Simulation experiments demonstrated that FSVC can improve the schedulability and resource utilization effectively in virtualized clouds.
Key words: virtualized cloud; fault-tolerant scheduling; primary-backup approach; real-time system

1 引言

云计算是一种动态提供计算资源的新型计算模式[1]。它通常依托于采用虚拟化技术的数据中心，以实现资源的动态整合与环境隔离。在各个 IT 企业的大力推动下（如 Google、IBM、Microsoft 等），云计算的应用领域在不断扩大[2]。值得注意的是，一些实时系统也被部署到了云平台中。这种系统的正确性不仅取决于其计算结果，还取决于产生结果的时间[3]，不能及时地反馈结果与产生错误结果一样糟糕，甚至可能导致人员伤亡、财产损失等灾难性的后果。

云计算使灵活、按需地使用大规模计算资源成为现实，但同时也带来了一个亟待解决的问题——增大了系统失效的可能性。文献[4]指出，在一个由 10 000 台高可靠性服务器构成的系统中，平均每天就有一个服务器失效。考虑到云服务提供商所使用的是大量廉价的商用计算机，这进一步增大了资源失效的可能性。另一方面，由于实时系统的安全关键性，在资源失效的情况下，系统仍能正常地提供服务就显得尤为重要。由此，产生了一个非常重要的问题，云平台中的容错问题。

收稿日期：2013-09-08；**修回日期**：2014-01-20
基金项目：高等学校博士学科点专项科研基金资助项目（20134307110029）；国家自然科学基金资助项目（61104180）；西南电子电信技术研究室公开课题基金资助项目（2013001）
Foundation Items：Specialized Research Fund for the Doctoral Program of Higher Education (20134307110029)；The National Natural Science Foundation of China (61104180)；Public Project of Southwest Inst. of Electron. & Telecom. Technology (2013001)

为了解决这一问题，容错技术被广泛地研究与部署。Okorafor 提出了一种虚拟机层面的检查点与回滚机制[5]，但该机制需要大量额外的存储空间与高效的恢复技术。MapReduce 为了解决 worker 失效的问题，受影响的任务被重置为初始状态并调度到其他正常 worker 上[6]，然而这不可避免地会导致服务质量下降，以致在执行实时任务时错过任务的截止期。大量证据表明，调度是一种在现有系统的基础上可以有效提高系统性能的方法，而容错调度通过复制技术将同一任务的多个版本分配到不同的计算实体上以实现容错[7]。近来，Zheng 研究了在 MapReducek 框架下如何通过容错调度提高系统性能，但他主要关注了数据密集型的任务，并没有考虑计算时间对任务完成情况的影响[8]。Antony 等人提出了一种云环境下考虑负载均衡的容错调度算法，但该算法没有考虑实时任务的截止期要求，也没有充分考虑如何提高系统资源利用率[9]。Plankensteiner 等人将减少任务计算时间与提高系统资源利用率作为目标，提出了一种综合复制技术与再提交技术的容错调度算法，但该算法只能适用于软实时任务，无法适应硬实时任务对截止期的严格要求[10]。此外，许多学者研究了如何在传统的分布式系统中(如集群、网格等)进行容错调度[3,11]。其中，基于主副版本（PB, primary/backup）技术是一种重要的方法。在该方法中，每个任务有 2 个版本，即主版本与副版本，分别被调度到 2 个不同的计算实体上以实现容错，但该方法增加了资源的使用，降低了系统的可调度性。为了缓解这个矛盾，Ghosh 等人提出了释放和重叠 2 种技术。释放是指在成功完成一个任务的主版本后，对应副版本将被删除以释放其占用的资源，而重叠是指在同一计算实体的同一时间段内多个副版本可以相互重叠[11]。基于这 2 种技术，许多学者研究了如何调度主副版本的算法。由于多任务调度问题是一个 NP 完全问题，因此在实际应用中一般采用启发式算法来进行调度[12]。针对同构系统，Oh 等人提出了一种可以容忍单处理器失效的静态容错算法[13]，Qin 等人提出了一种在异构系统中，针对相互依赖的任务容错调度算法[3]，但这 2 种方法本质上是静态的，不适合进行动态调度。在很多情况下，任务是非周期的，到达时间、截止期都不能预先获知，这就需要一种动态的调度方法。Zheng 等人设计了 MRC-ECT 和 MCT-LRC 2 种算法，调度独立任务与相互依赖的任务，分别达到最小化复制成本和最小化完成时间的目的[14]。Zhu 等人分别采用经典的 ASAP（as soon as possible）与 ALAP（as late as possible）策略来动态地调度主版本与副版本[13]。虽然这些方法在传统的分布式系统中有良好的表现，但对于云计算，这些方法都存在一个重要的缺陷，即没有考虑云的一个关键特征——虚拟化。

虚拟化技术将一台物理主机动态地划分为多个可独立提供计算服务的虚拟机[15]，而虚拟机迁移技术可以进一步提高资源的利用率，但这种技术使容错调度变得更具挑战性。首先，任务是被调度到虚拟机上而非主机上，一个主机的失效会导致多个虚拟机失效，进而致使多个虚拟机上的任务不能执行，这增加了调度的难度。其次，为了提高资源的利用率，算法中必须包含虚拟机迁移技术，由此就要研究迁移的约束以满足容错的要求，这又增加了设计算法的难度。

现有的研究成果均未能很好地解决上述问题。本文的工作就立足于考虑云平台虚拟化的特点，针对单个主机永久或暂时失效的情况，提出一种虚拟化云平台中基于主副版本的容错调度算法，主要贡献如下。

1) 分析了云计算中副版本重叠应满足的约束。

2) 首次分析了在满足容错要求的前提下，虚拟机迁移应满足的约束。

3) 考虑云的特点，提出了一种两阶段调度策略，以进一步提高算法性能。

4) 提出了一种基于主副版本方法的容错调度算法(FSVC, fault-tolerant scheduling algorithm in virtualized clouds)，并进行了仿真实验，评估了该算法的性能。

2 系统模型

2.1 失效模型

本文中失效模型的假设与文献[3]类似。

1) 在任一时间，至多只有一台主机失效。一台主机失效后，主版本在该主机上的任务可在另一个主机失效之前由其副版本成功完成。

2) 失效既可以是暂时的也可以是永久的，但各个失效相互独立，一台主机的失效不会影响其他主机。

3) 系统中存在一个失效的探测机制，可以提供失效的信息，新任务不会被调度到已失效的主机上。

需要注意的是，针对多个主机同时失效的情

况，该失效模型可以通过下面 2 个步骤做扩展。首先，将云平台中的主机分为若干个小组；然后，在每个主机小组内适用上述失效模型。于是，类似于文献[11]，在各个小组内采用本文所提出的容错机制，以解决多主机失效的情况。

2.2 任务模型

集合 $T=\{t_1,t_2,\cdots,t_n\}$ 表示一组非抢占、非周期的任务。每个任务 t_i 有 3 个属性：到达时间 a_i、截止期 d_i 与计算量 cs_i，计算量 cs_i 用百万指令数（MI，million instructions）衡量。每个任务 t_i 有 2 个版本 t_i^P 与 t_i^B，被调度到两台主机上。对于任务 $t_i\in T$，s_i^P 与 f_i^P 表示 t_i 主版本的开始时间与结束时间，s_i^B 与 f_i^B 表示副版本的开始与结束时间。

无限集 $H=\{h_1,h_2,\cdots\}$ 表示虚拟化云平台中无限的主机。每个主机有一定的处理能力 P_i，P_i 用每秒百万指令数（MIPS，million instructions per second）衡量。要注意的是，虽然云中的主机数量是无限的，但活动主机的数量是有限的。集合 $H_a=\{h_1,h_2,\cdots,h_{|H_a|}\}$ 表示 H 中的活动主机数，而 $H-H_a$ 是关闭的主机。每个主机 h_i 上有多个虚拟机，用集合 $V_i=\{v_{i1},v_{i2},\cdots,v_{i|V_i|}\}$ 表示，每个虚拟机有不同的处理能力 P_{ij}。对于主机 h_i 上的虚拟机，其处理能力满足约束 $\sum_{j=1}^{|V_i|}P_{ij}\leqslant P_i$。

副版本有 2 种执行模式，即主动副版本与被动副版本。前者允许副版本与主版本同时执行，后者表示副版本仅在主版本出错之后才开始执行。本文根据任务主版本的时间裕度，自适应地决定副版本的执行模式。若副版本 t_i^B 被调度到虚拟机 v_{kl} 上，则 t_i^B 的执行模式 $st(t_i^B)$ 表示为

$$st(t_i^B)=\begin{cases}\text{passive}, f_i^P+e_{kl}(t_i)\leqslant d_i\\ \text{active}, f_i^P+e_{kl}(t_i)>d_i\end{cases}\tag{1}$$

为了尽可能地减少副版本占用的资源，引入释放技术，删除已成功完成的主版本所对应的副版本。因此，副版本的实际执行时间不仅与它的模式有关，还与对应主版本是否成功完成有关。副版本 t_i^B 在虚拟机 v_{kl} 上的实际执行时间 $ae_{kl}(t_i^B)$ 由式(2)确定：

$$ae_{kl}(t_i^B)=\begin{cases}0, t_i^P\text{ succeeds}\wedge st(t_i^B)=\text{passive}\\ f_i^B-s_i^B, t_i^P\text{ succeeds}\wedge st(t_i^B)=\text{active}\\ e_{kl}(t_i), t_i^P\text{ fails}\end{cases}\tag{2}$$

3 云平台中的约束

3.1 副版本重叠的约束

在传统的系统中，2 个相互独立的任务只要它们的主版本不在一个计算实体上，它们的副版本就可以重叠。但是，在云平台中，情况就要变得更为复杂，本节将明确在云平台中，2 个副版本相互重叠所应满足的约束。

与传统的系统不同，云平台中 2 个副版本可以相互重叠的基本前提是它们对应的主版本没有被调度到同一台主机的虚拟机上。

定理 1　如果 $\text{host}(t_i^P)=\text{host}(t_j^P)$，那么 t_i^B 与 t_j^B 不能重叠，无论 $\text{vm}(t_i^P)$ 是否等于 $\text{vm}(t_j^P)$。

证明　假设 $\text{host}(t_i^P)=\text{host}(t_j^P)$，且 t_i^B 与 t_j^B 可以重叠。不妨设 $\text{host}(t_i^P)=\text{host}(t_j^P)=h_1$，当 h_1 失效时，h_1 上的所有虚拟机均失效，致使调度到这些虚拟机上的所有任务版本均出错，包括 t_i^P 与 t_j^P。因此，t_i^B 与 t_j^B 都必须执行，t_i^B 与 t_j^B 又有重叠，出现同一虚拟机上 t_i^B 与 t_j^B 同时执行，相互冲突。所以，假设不成立。

不失一般性，假设任务 t_i 的 2 个版本都已经被调度，任务 t_j 是新到达将要被调度的任务。下面根据 t_i^B 的执行模式，分 2 种情况讨论副版本重叠时应满足的其他约束条件。

1) t_i^B 采用被动执行模式

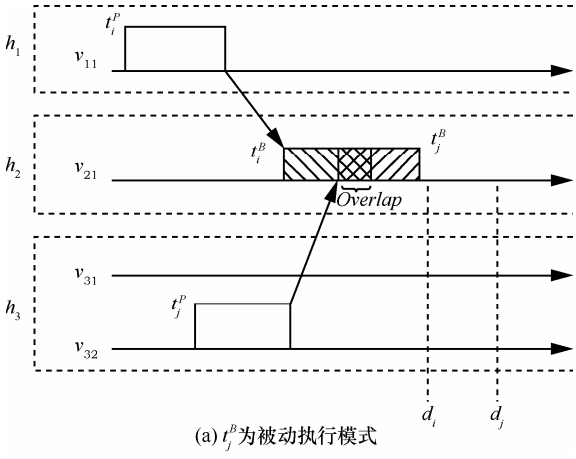
图 1 给出了这种情况下的 2 个实例。图 1(a)中 t_j^B 采用被动执行模式，且 $\text{host}(t_i^P)\neq\text{host}(t_j^P)$。根据 2.1 节中的假设，任一时间只会有一台主机失效，显然 t_i^B 与 t_j^B 可以重叠。令 $LST_{kl}(t_j^B)$ 表示 t_j^B 在虚拟机 v_{kl} 上的最晚开始时间，则

$$LST_{kl}(t_j^B)=d_j-e_{kl}(t_j)\tag{3}$$

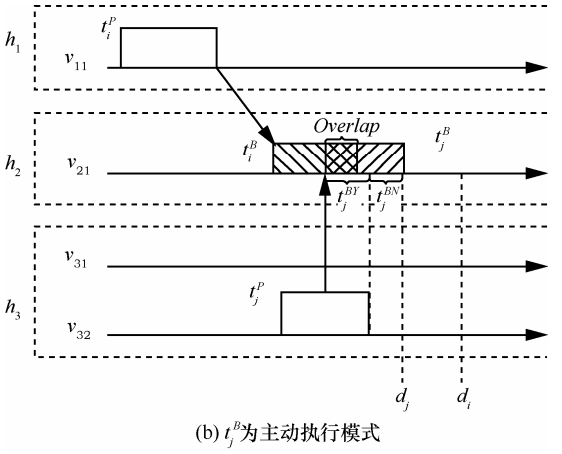
图 1(b)中 t_j^B 为主动执行模式。t_j^B 的执行可以分为 2 部分：t_j^{BY} 与 t_j^{BN}，t_j^{BY} 为与主版本同时执行的部分，执行 t_j^{BN} 部分时主版本已完成。在这种情况下，t_j^B 不能与 t_i^B 重叠。

定理 2　如果 $st(t_i^B)=\text{passive}$，$st(t_j^B)=\text{active}$，那么 t_i^B 与 t_j^B 不可以重叠。

证明　假设 t_i^B 与 t_j^B 重叠：若 $\text{host}(t_i^B)$ 失效，则 t_i^B 必须执行，而 t_j^B 是主动执行模式，t_j^{BY} 也必须执行，出现同一台虚拟机上 t_i^B 与 t_j^{BY} 同时执行，相互冲突。所以，假设不成立。

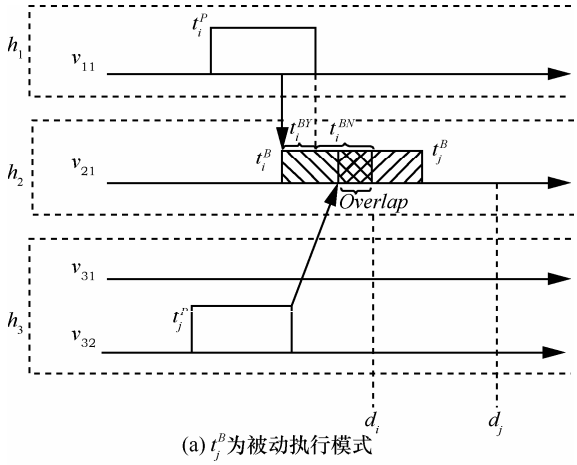


(a) t_j^B为被动执行模式

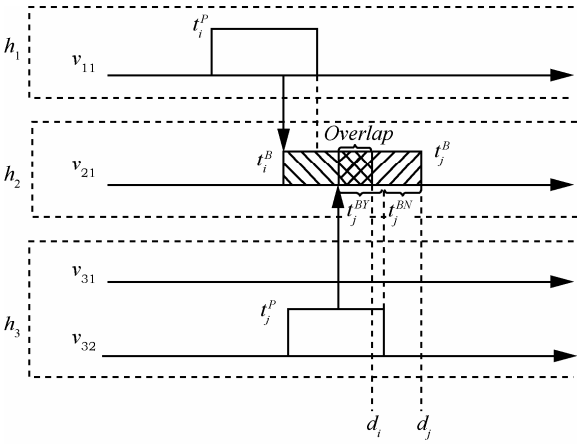


(b) t_j^B为主动执行模式

图 1　t_i^B 采用被动执行模式的实例



(a) t_j^B为被动执行模式



(b) t_j^B为主动执行模式

图 2　t_i^B 采用主动执行模式的实例

2)　t_i^B 采用主动执行模式

图 2 中 t_i^B 均采用了主动执行模式，t_i^{BY} 为与主版本 t_i^P 同时执行的部分，t_i^{BN} 部分没有与主版本同时执行。图 2(a)中，t_j^B 是被动执行模式，在满足式(3)所述的截止期约束的同时，t_j^B 的开始时间还不能早于 f_i^P。

定理 3　如果 $st(t_i^B)=\text{active}$，$st(t_j^B)=\text{passive}$，且 t_i^B 与 t_j^B 重叠，那么

$$LST_{kl}(t_j^B) \geqslant f_i^P \tag{4}$$

证明　假设 $LST_{kl}(t_j^B)$ 早于 f_i^P，那么 t_j^B 与 t_i^{BY} 重叠。由于 t_i^B 是主动执行模式，t_i^{BY} 必须与 t_i^P 同时执行。若 $\text{host}(t_j^P)$ 失效，则 t_j^B 必须执行，出现同一虚拟机上 t_i^{BY} 与 t_j^B 同时执行，相互冲突。所以，假设不成立。

图 2(b)中，t_j^B 是主动执行模式，此时 t_i^B 不能与 t_j^B 重叠，证明过程与定理 2 类似，若 $\text{host}(t_i^B)$ 失效，t_i^B 与 t_j^B 重叠会导致 t_i^B 与 t_j^{BY} 相互冲突。

3.2　虚拟机迁移的约束

副版本重叠技术通过缩短副版本实际占用的计算时间来提高资源利用率，但它仅在虚拟机层面上发挥作用。依靠虚拟机迁移技术，可以在主机层面上进一步提高资源利用率。为了达到容错的要求，下面分析虚拟机迁移时所要满足的约束。

定理 4　令 fH_{kl} 表示虚拟机 v_{kl} 不能迁往的主机集合。则 $fH_{kl}=\{\text{host}(t_i^P)\mid \forall t_i\in T, \text{vm}(t_i^B)=v_{kl}\} \cup \{\text{host}(t_i^B)\mid \forall t_i\in T, \text{vm}(t_i^P)=v_{kl}\}$，$v_{kl}$ 不能迁移到 h_j 上，$\forall h_j \in fH_{kl}$。

证明　假设 v_{kl} 迁移到 h_j 上，$\forall h_j \in fH_{kl}$，将使主版本与副版本被调度到同一台主机的虚拟机上，当该主机失效时，会导致主版本与副版本同时出错，显然不满足容错的要求。所以，假设不成立。实例如图 3 所示。

定理 5　$\forall t_i \in T$，$\text{vm}(t_i^P)=v_{kl}$，如果 t_i^B 与 t_j^B 重叠，那么 v_{kl} 不能迁移到主机 $\text{host}(t_j^P)$ 上。

证明　假设 v_{kl} 迁移到主机 host(t_j^P) 上，那么相互重叠的副版本对应主版本被调度到同一台主机的虚拟机上，违背定理 1 的约束。所以，假设不成立。实例如图 4 所示。

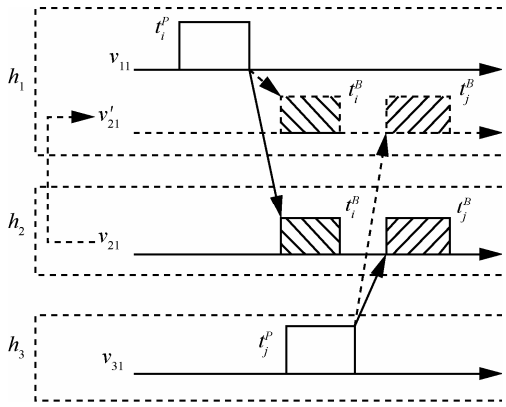


图 3　定理 4 的实例

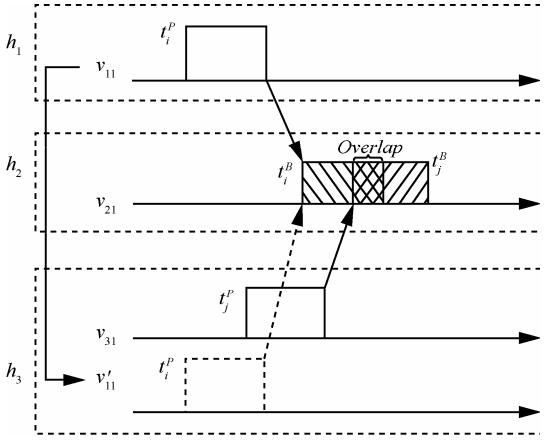


图 4　定理 5 的实例

4　容错调度算法

本节设计一种虚拟化云平台中的容错调度算法 FSVC，该方法利用了副版本重叠与虚拟机迁移技术，以达到在满足容错要求的同时提高资源利用率的目的。当一个新任务到达云后，系统首先调度主版本，然后调度副版本。主版本或副版本任意一个被拒绝，这个任务都会被拒绝。

4.1　主版本调度算法

为了给副版本留下足够的时间裕度，主版本应尽早完成。同时，根据定理 1，2 个副版本能重叠的基本前提是它们的主版本不在同一台主机上。因此，主版本不能集中调度在若干台主机上，这会使副版本之间的重叠不可行，而降低资源利用率。所以，主版本的调度要达到下面 2 个目标。

1) 主版本的完成时间尽可能早。

2) 主版本被均匀地调度到各活动主机上。

云计算与传统计算模式的一个重要不同点就在于任务是被调度到虚拟机上，而非主机上。然而，为了达到上述 2 个目标，在进行调度时主机的状态也必须考虑在内。为此，本文提出一种两阶段策略。

首先，确定候选主机集合。主版本数量较少的主机被选为候选主机。本文的算法中，主版本数最少的前 α% 个主机被选为候选主机；其次，选择分配给主版本的虚拟机。在候选主机上的所有虚拟机中，选择一个虚拟机，在该虚拟机上，主版本的完成时间最早。

通过这个策略，新到达任务的主版本会被优先调度到主版本数较少的主机上以实现主版本的均匀分配。α 越小，主版本的分布越均匀，但过小的 α 会使可选择的虚拟机数量过少，而延后主版本的开始时间。一个合适的 α 可以在主版本尽早完成与均匀分配之间取得平衡。

如果现有的虚拟机都不能容纳主版本 t_i^P，可以新创建一个虚拟机。新虚拟机的处理能力 P_{new} 满足

$$a_i + cs_i / P_{\text{new}} + delay = d_i \tag{5}$$

其中，*delay* 表示为开启主机、创建虚拟机而预留的延时。本文的算法首先尝试将新虚拟机放置到活动主机上，若没有合适的活动主机，则进行虚拟机迁移，整合现有虚拟机，为新虚拟机空出主机资源。如果仍然没有合适的活动主机，那么从关闭的主机 $H - H_a$ 中开启一台主机，将新虚拟机放置到新的活动主机上。算法 1 给出了 FSVC 中调度主版本的伪代码。

算法 1　主版本调度伪代码

1) Sort H_a in an increasing order by count of scheduled primaries;

2) $H_{\text{candidate}}$←top α% hosts in H_a;

3) *find*←FALSE; *EFT*←+∞; *v*←*NULL*;

4) **While** !all hosts in H_a has been scanned **do**

5)　**for** each h_k in $H_{\text{candidate}}$ **do**

6)　**for** each v_{kl} in h_k.*VmList* **do**

7)　Calculate the earliest finish time $EFT_{kl}(t_i^P)$;

8)　**if** $EFT_{kl}(t_i^P) \leqslant d_i$ **then**

9)　*find*←TRUE;

10)　**if** $EFT_{kl}(t_i^P) \leqslant EFT$ **then**

11)　$EFT \leftarrow EFT_{kl}(t_i^P)$; $v \leftarrow v_{kl}$;

12)　**end if**

13)　　**end if**
14)　**end for**
15)　**end for**
16)　**if** *find*==FALSE **then**
17)　$H_{\text{candidate}}$←next top α% hosts in H_a;
18)　**else**
19)　break;
20)　**end if**
21) **end while**
22) **if** *find*==FALSE **then**
23)　Create *newVm* with processing power P_{new};
24)　**for** h_k in H_a **do**
25)　**if** h_k can accommodate *newVm* **then**
26)　Allocate *newVm* to h_k;
27)　*v*←*newVm*;　*find*←TRUE;
28)　break;
29)　**end if**
30)　**end for**
31)　**if** *find*==FALSE **then**
32)　*sourceHost*←**vmMigration**(P_{new});
33)　**if** *sourceHost*≠*NULL* **then**
34)　Allocate *newVm* to *sourceHost*;
35)　*v*←*newVm*;　*find*←TRUE;
36)　**end if**
37)　**end if**
38) **end if**
39) **if** *find*==FALSE **then**
40)　Turn on a host h_{new} in H-H_a;
41)　**if** MIPS of h_{new} satisfies P_{new} **then**
42)　Allocate *newVm* to h_{new};
43)　*v*←*newVm*;　*find*←TRUE;
44)　**end if**
45) **end if**
46) **if** *find*==TRUE **then**
47)　Allocate t_i^P to *v*;
48) **else**
49)　Reject t_i^P;
50) **end if**

算法 1 中包含了虚拟机迁移技术以提高资源利用率。这里再次采用两阶段策略，以达到虚拟机迁移在整合虚拟机的同时，使主版本的分布更均匀，并尽可能减小对已调度版本的影响。

首先，选择一台主版本数最多的主机。迁移该主机上的虚拟机，减少主版本数；其次，选择这台主机上主版本数量最少的虚拟机进行迁移，达到减小对已调度版本的影响。虚拟机迁移的伪代码如算法 2 所示。

算法 2　Function vmMigration(P_{new})

1) **Input:** P_{new}　　**Output:** *sourceHost*;
2) Sort H_a in a decreasing order by count of scheduled primaries;
3) *sourceHost*←NULL;
4) **for** each h_i in H_a **do**
5)　Sort h_i.*vmList* in increasing order by number of scheduled primaries;
6)　*nowP*←0;　*migrationList*←∅;
7)　**for** each v_j in h_i.*vmList* **do**
8)　**for** h_k in inverted H_a **do**
9)　**if** v_j's migration to h_k meets the VM migration constraints && h_k can accommodate v_j **then**
10)　*nowP*←*nowP*+v_j.*power*;
11)　*migrationList*.add{v_j,h_k};
12)　break;
13)　**end if**
14)　**end for**
15)　**if** *nowP*≥P_{new} **then**
16)　break;
17)　**end if**;
18)　**end for**
19)　**if** *nowP*≥P_{new} **then**
20)　*sourceHost*←h_i;
21)　Migrate VMs in *migrationList*;
22)　break;
23)　**end if**
24) **end for**

下面分析主版本调度算法的时间复杂度。令 N_a 表示云平台中活动主机的总数，N_{vm} 表示各个活动主机上虚拟机个数的最大值，N_{tw} 表示所有虚拟机上等待执行任务个数的最大值。为了在已有的资源中确定一台合适的虚拟机，算法 1 中 1)~21)行在最好情况下(即在第一组候选主机中就找到合适的虚拟机)时间复杂度为 $O(\alpha N_a N_{vm} N_{tw})$，而在最差的情况下(即探寻了所有的活动主机)为 $O(N_a N_{vm} N_{tw})$。函数 vmMigration(P_{new})的时间复杂度为 $O(N_{vm} N_a^2)$，从而 22)~50)行新增一台虚拟机的时间复杂度为

$O(N_a+N_{vm}N_a^2+1)=O(N_{vm}N_a^2)$。因此，在最好的情况下，主版本调度算法的时间复杂度为 $O(\alpha N_a N_{vm} N_{tw})$，最差的情况下为 $O(N_a N_{vm} N_{tw}+N_{vm}N_a^2)$。值得注意的是，通常情况下，并不需要探寻所有主机或新增资源，由此可以发现引入两阶段策略后，该算法相比于经典的 ASAP 策略，不仅可以使主版本的分布更均匀，还可以降低算法的复杂度。

4.2　副版本调度算法

在设计算法之前，首先引入文献[11]中介绍的一个概念，复制成本：

$$C_{kl}^{R}(t_i^B)=\frac{e_{kl}(t_i^B)-t_{kl}^{O}(t_i^B)}{e_{kl}(t_i^B)} \tag{6}$$

其中，$t_{kl}^{O}(t_i^B)$ 表示在虚拟机 v_{kl} 上 t_i^B 与其他副版本重叠的计算时间。复制成本表示除了与现有版本重叠的部分，该副版本事实上所占用的计算时间与执行时间的比值。一个小的复制成本意味着除了重叠的部分，这个副版本需要计算时间比较小。换句话说，实际的资源占用量比较少。为了提高资源利用率，调度副版本时要达到 2 个目标。

1) 副版本应尽可能地采用被动模式。

2) 副版本的复制成本要尽可能小。

基于这 2 个目标，本算法使用了 ALAP 策略，在使副版本的开始时间最迟的前提下，使副版本的复制成本最小。算法 3 给出了 FSVC 中调度副版本的伪代码。

算法 3　副版本调度伪代码

1) Sort H_a in the increasing order by the remaining MIPS;

2) *find*←FALSE;

3) **for** each h_k in H_a except $host(t_i^P)$ **do**

4)　**for** each v_{kl} in $h_k.vmList$ **do**

5)　Calculate the latest start time $LST_{kl}(t_i^B)$;

6)　**if** $LST_{kl}(t_i^B)\leqslant d_i$ **then**

7)　*find*←TRUE;

8)　Calculate the replication cost $C_{kl}^{R}(t_i^B)$;

9)　**if** $LST_{kl}(t_i^B)\geqslant d_i \,||\, LST_{kl}(t_i^B)=LST \,\&\&\, C_{kl}^{R}(t_i^B)<C$ **then**

10)　$LST\leftarrow LST_{kl}(t_i^B)$;　$C\leftarrow C_{kl}^{R}(t_i^B)$;

11)　$v\leftarrow v_{kl}$;

12）　**end if**

13)　**end if**

14)　**end for**

15) **end for**

16) The following part is similar to line 22)~50) in Algorithm 1.

5　性能分析

本节通过大量的仿真实验来检验 FSVC 的性能。将 FSVC 与 2 种基准算法: SFSVC（single-phase-FSVC），NMFSVC（non- migration-FSVC）作比较，以检验 FSVC 中所采用的各种技术的效果。SFSVC 中，调度主版本时不考虑主机上主版本的数量，直接在所有活动主机中选择虚拟机（即经典的 ASAP 与 ALAP 策略[13]）。NMFSVC 与 FSVC 的区别在于前者没有采用虚拟机迁移技术。

本文从以下 3 个指标来比较算法的性能。

1) 完成率（GR, guarantee ratio）：成功调度的任务占所有提交任务的百分比。

2) 活动主机数（AHC, active host count）：整个调度过程中，云中的活动主机总数。

3) 平均单任务主机数（HPT, hosts per task）：活动主机数与成功调度任务数的比值。

5.1　模拟方法和参数

本文的实验采用 CloudSim 模拟云平台。每台主机有一个 CPU，为体现处理能力的异构性，参数可以为 1 000、1 500 或 2 000 MIPS。每个虚拟机需要一个性能为 200、300 或 400 MIPS 的核心。本文设定开启一台主机和创建一个虚拟机的时间分别为 90 s 和 15 s。实验中有 10 000 个任务到达云，假设它们到达服从平均间隔时间为 $1/\lambda$ 的泊松分布，$1/\lambda$ 是 $[1/\lambda_0, 1/\lambda_0+2]$ 之间的均匀分布。任务的计算量在 $[1\times10^5, 2\times10^5]$ 范围内均匀分布。截止期为 $d_i=a_i+deadlineTime$，$deadlineTime$ 服从均匀分布，$U(baseDeadline, 4baseDeadline)$。表 1 给出了参数和取值，每个实验重复进行 50 次。

表 1　　实验参数设定

参数	取值(固定值)—(最小值，最大值，步长)
计算量(10^5 MI)	([1,2])
间隔时间/s	(2)—(0,16,2)
$baseDeadline$(10^2)/s	(4)—(1.5,6.5,0.5)

5.2　参数 α 对性能的影响

FSVC 对现有调度算法的一个重要改进在于引入了两阶段策略，在该策略下，首先选择候选主机，

然后从候选主机中选择合适的虚拟机。参数 α 的值决定了候选主机范围的大小，会影响算法的性能。本实验将 α 的值从 0.1 变化到 1，设定表 1 中的参数值为固定值。当 α=1 时，所有的活动主机都被选为候选主机，FSVC 也就退化为 SFSVC。图 5 显示了 FSVC 与 SFSVC 的性能。

从图 5(a)中可以看到，FSVC 的完成率要高于 SFSVC。这个优势来源于两阶段策略使主版本均匀分配在各个活动主机上，从而使更多的副版本可以重叠，提高了系统的可调度性。另外，还能看到随着 α 的变化，FSVC 的完成率也在变化。当 α 从 0.1 变化到 0.3 时，完成率增长，在 0.3 附近达到极值。接着随着 α 的值接近于 1，FSVC 退化为 SFSVC。原因在于一个过小的 α 会使候选主机的范围太小，延后主版本的开始时间，影响副版本的调度。而当 α 接近于 1 时，候选主机的范围又变得太大，使两阶段策略退化为单阶段策略。

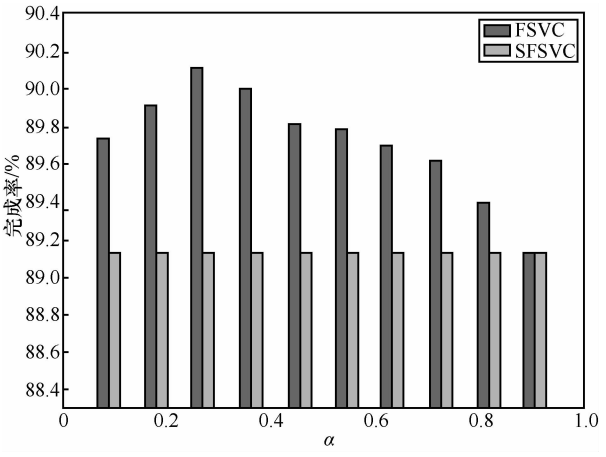
图 5(b)显示 FSVC 的活动主机数随着 α 的增长而增长。但 FSVC 总比 SFSVC 的性能好。原因与图 5(a)类似。较小的 α 可以使更多的副版本重叠，从而提高系统的资源利用率。当 α 变大时，副版本要占用更多的计算资源。因此，需要更多的活动主机来容纳这些版本。

从图 5(c)中可以看到平均单任务主机数随着 α 的增长而增长。当 α 较小时，平均单任务主机数也较少。一个有趣的现象在于平均单任务主机数的变化趋势与活动主机数类似。这意味着 FSVC 相比 SFSVC 可以不减少任务完成数的情况下显著减少使用计算资源。

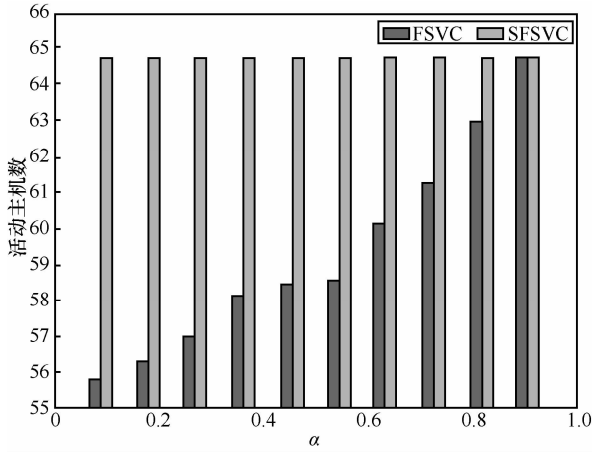
上述实验结果表明 α 的值对算法的性能有重要影响。为了在完成率与资源利用率之间达到一个较好的平衡，在下面的实验中设定 α=0.2。

5.3　任务到达率对性能的影响

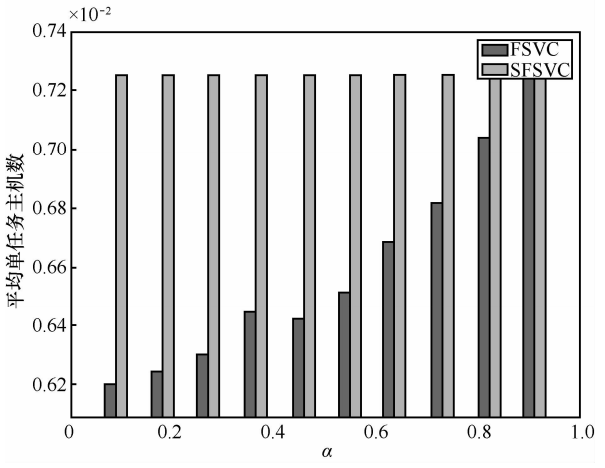
图 6(a)给出当间隔时间 $1/\lambda$ 变化时，FSVC、SFSVC 与 NMFSVC 都有一个较高的任务完成率（89%以上），这是因为云计算模式下，可以无限量地按需提供资源。然而，大量几乎同时到达的任务会使系统过载，由于新开主机需要额外的开机时间，每个任务又有截止期限制，因此会导致一些任务被拒绝。另外，在 *Interval Time* 为 2 和 4 时，NMFSVC 明显比 FSVC 表现更优。其原因在于 NMFSVC 倾向于打开更多的主机，这可以在后续任务达到时缓解系统的过载。



(a) 完成率随 α 的变化



(b) 活动主机数随 α 的变化

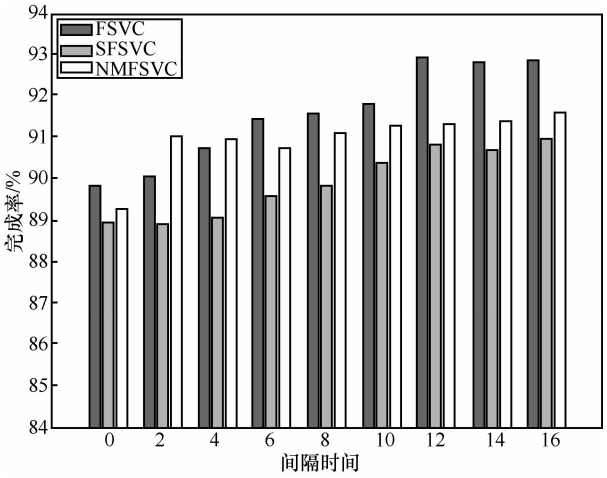


(c) 平均单任务主机数随 α 的变化

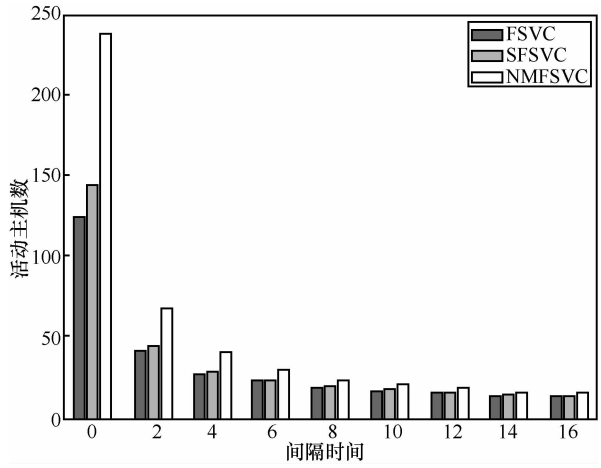
图 5　参数 α 对性能的影响

图 6(b)说明 FSVC 与 SFSVC、NMFSVC 相比需要的活动主机更少。原因与图 5(a)类似。另外，图 6(b)中，NMFSVC 需要 2 倍 FSVC 的活动主机数，而 SFSVC 只需要比 FSVC 多 20%。这说明在节约活动主机数上，虚拟机迁移技术比两阶段策

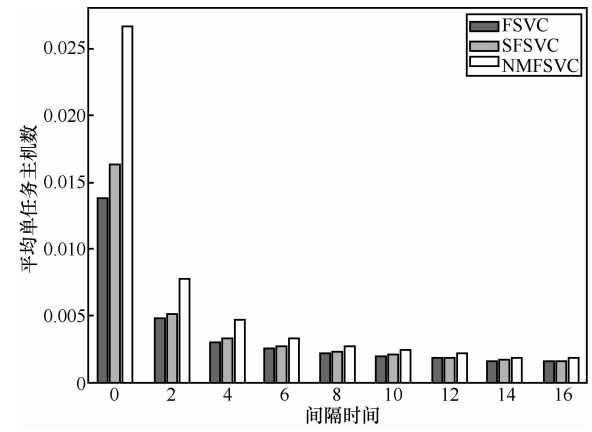
略要更为有效。原因在于两阶段策略通过使更多的副版本相互重叠来减少占用的虚拟机资源，这仅仅直接减少了虚拟机的数量。而虚拟机迁移技术在主机层面上发挥作用，可以直接减少活动主机数。



(a) 完成率随间隔时间的变化



(b) 活动主机数随间隔时间的变化



(c) 平均单任务主机数随间隔时间的变化

图 6　任务到达率对性能的影响

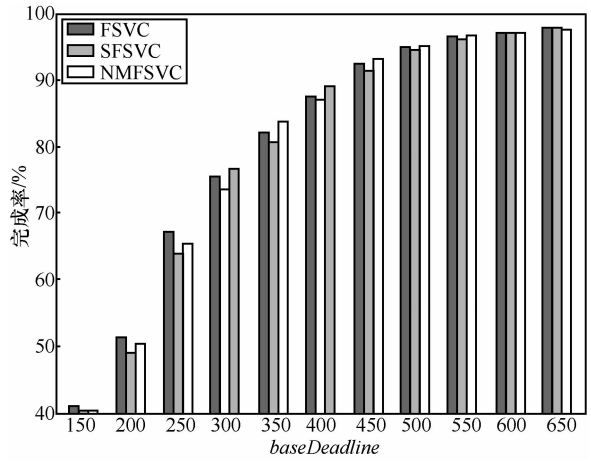
FSVC、SFSVC 与 NMFSVC 的平均单任务主机数如图 6(c)所示。在区间 0 到 4 内，FSVC 的性能分别比 SFSVC 与 NMFSVC 提高了 14.3%和 91.8%。这个结果进一步证明虚拟机迁移技术在提高资源利用率上有巨大优势。

5.4　截止期对性能的影响

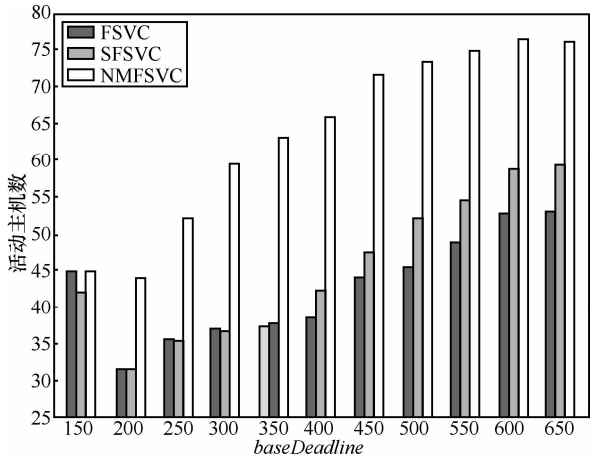
在图 7(a)中可以看到，当 *baseDeadline* 增加（即截止期变宽松）时，3 种算法的完成率都显著提高。这是因为在截止期非常紧张时，额外的开机时间会使新开主机也变得没有作用。当截止期足够宽裕时，几乎所有的任务都可以被成功调度。另外，图 7(a)显示 SFSVC 的完成率最小，在区间 300 到 500 中，NMFSVC 的完成率甚至比 FSVC 的大。有 2 个原因：其一，单阶段策略使副版本难以重叠，所以更多的任务无法被成功调度；其二，当 *baseDeadline* 在区间 300 到 500 内变化时，截止期对于调度副版本而言仍然比较紧张，但这个时间足够去开启新的主机，因此相比虚拟机迁移，新开主机更能提高完成率。

在图 7(b)中，当 *baseDeadline* 为 150 时，截止期非常紧张，以至于 3 种算法都要开启大量主机来容纳这些任务。当截止期变得稍微宽松一些后，时间余量已经能满足虚拟机迁移的要求，FSVC 与 SFSVC 的活动主机数大幅下降。当截止期进一步宽松时，活动主机数上升，这是因为截止期变宽松时，系统能接受更多的任务，需要更多的主机来容纳这些任务。另外，在区间 150 到 350 中，FSVC 与 SFSVC 几乎有相同的活动主机数。这是因为两阶段策略减少了可选虚拟机的数量，会推迟主版本的开始时间。同时，紧张的截止期使副版本必须采用主动模式，甚至与主版本同时开始，这使得副版本的重叠更加困难，必须使用更多资源。因此 FSVC 在这个区间内性能与 SFSVC 相似。

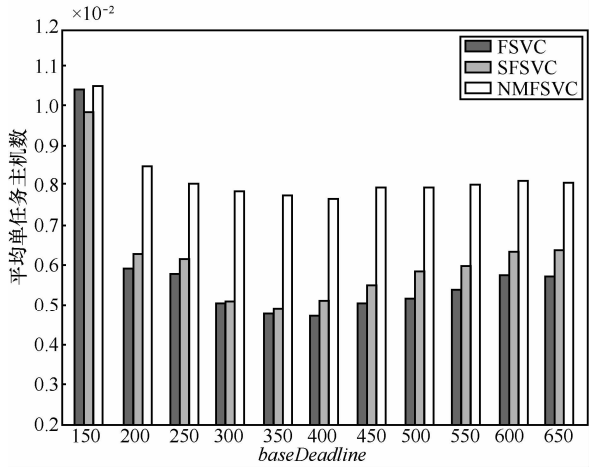
图 7(c)中显示，FSVC 的平均单任务主机数要比 SFSVC 和 NMFSVC 的小。另外，当 *baseDeadline* 大于 400 时，FSVC 与 SFSVC 都出现缓慢的上升。这是因为，当 *baseDeadline* 在 200 到 400 内变化时，虚拟机迁移技术可以提高资源利用率的优势。然而，当截止期进一步宽松时，更多的任务被接受，必须开启更多的主机，致使一些主机上的计算能力不能得到充分利用而空闲，效率变低，平均单任务主机数上升。



(a) 完成率随 *baseDeadline* 的变化



(b) 活动主机数随 *baseDeadline* 的变化



(c) 平均单任务主机数随 *baseDeadline* 的变化

图 7　截止期对性能的影响

6　结束语

本文研究了虚拟化云平台中独立任务的实时容错调度问题，其调度的目标是在满足容错要求的前提下，提高系统可调度性与资源利用率。本文在主副版本方法的基础上，考虑虚拟化云平台的特点，分析明确了副版本重叠与虚拟机迁移的约束，通过使用这 2 种技术来提高调度的性能。另外，本文采用了一种两阶段策略来进一步提高性能。大量的仿真实验表明本文的算法可以显著提高系统可调度性与资源利用率。

下一步的研究工作主要包括 3 个方面：1) 扩展容错调度模型并提出相应算法，从而可以容忍多个主机失效；2) 考虑通信时延等云平台的其他特性，使模型更为精确；3) 在实际系统中部署该方法，检验算法的性能。

参考文献：

[1] BARHAM P. DRAGOVIC B, FRASER K, *et al*. Xen and the art of virtualization[J]. ACM SIGOPS Operating Systems Review, 2003, 37(5): 164-177.

[2] Above the Clouds: A Berkeley View of Cloud Computing[R]. USA: University of California, Berkeley, 2009.

[3] QIN X, JIANG H. A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems[J]. Parallel Computing, 2006, 32(5): 331-356.

[4] JERRREY D. Designs, Lessons and Advice From Building Large Distributed Systems[R]. Keynote from LADIS, 2009.

[5] OKORAFOR E. A fault-tolerant high performance cloud strategy for scientific computing[A]. Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)[C]. 2011. 1525-1532.

[6] JEFFREY D, SANJAY G. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[7] MANIMARAN G, MURTHY C S R. A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis[J]. IEEE Transactions on Parallel and Distributed Systems, 1998, 9(11): 1137-1152.

[8] ZHENG Q. Improving MapReduce fault tolerance in the cloud[A]. Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)[C]. 2010. 1-6.

[9] ANTONY S, ANTONY S, BEEGOM A, *et al*. Task scheduling algorithm with fault tolerance for cloud[A]. International Conference on Computing Sciences[C]. 2012. 180-182.

[10] PLANKENSTEINER K, PRODAN .R. Meeting soft deadlines in scientific workflows using resubmission impact[J]. IEEE Transactions on Parallel and Distributed Systems, 2012, 23(5): 890-901.

[11] GHOSH S, MELHEM R, MOSSÉ D. Fault-tolerance through scheduling algorithm for multiprocessor real-time systems and its analysis[J]. IEEE Transactions on Parallel and Distributed Systems, 1997, 8(3): 272-284.

（下转第 191 页）

[11] AHMED A A. An enhanced real-time routing protocol with load distribution for mobile wireless sensor networks[J]. Computer Networks, 2013, 57(6): 1459-1473.
[12] GAO S, ZHANG H K, DAS S K. Efficient data collection in wireless sensor networks with path-constrained mobile sinks[J]. IEEE Trans on Mobile Computing, 2011, 10(4): 592-608.
[13] 张希伟, 沈琳, 蒋益峰. 移动协助传感器网络中 sink 的路径优化策略[J].通信学报, 2013, 34(2): 85-93.
ZHANG X W, SHEN L, JIANG Y F. Optimizing path selection of mobile sink nodes in mobility-assistant WSN[J]. Journal on Communications, 2013, 34(2): 85-93.
[14] BETTSTETTER C, HARTENSTEIN H, PEREZ-COSTA X. Stochastic properties of the random waypoint mobility model[J]. Wireless Networks, 2004, 10(5): 555-567.
[15] BEHDANI B, YUN Y S, SMITH J C, *et al*. Decomposition algorithms for maximizing the lifetime of wireless sensor networks with mobile sinks[J]. Computers & Operations Research, 2012, 39(5): 1054-1061.
[16] LIU A F, REN J, LI Y. Design principles and improvement of cost function based energy aware routing algorithms for wireless sensor networks[J]. Computer Networks, 2012, 56(7): 1951-1967.
[17] TAO Y L, ZHANG Y B, JI Y S. Flow-balanced routing for multi-hop clustered wireless sensor networks[J]. Ad Hoc Networks, 2013, 11(1): 541-554.
[18] LIU T, PENG J, WANG X F, *et al*. Research on the energy hole problem based on non-uniform node distribution for wireless sensor networks[J]. KSII Transactions on Internet and Information Systems, 2012, 6(9): 2017-2036.

作者简介：

李慧杰（1989-），女，新疆乌苏人，四川大学硕士生，主要研究方向为无线传感器网络。

彭舰[通信作者]（1970-），男，四川成都人，四川大学教授，主要研究方向为无线传感器网络、分布式计算、人类动力学。E-mail:jianpeng@scu.edu.cn。

刘唐（1980-），男，四川乐山人，博士，四川师范大学副教授，主要研究方向为无线传感器网络。

（上接第 180 页）

[12] ZHU X M, QIN X, QIU M K. QoS-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters[J]. IEEE Transactions on Computers, 2011, 60(6): 800-812.
[13] OH Y, SON S H. Scheduling real-time tasks for dependability[J]. Journal of the Operational Research Society, 1997, 48(6): 629-639.
[14] ZHENG Q, VEERAVALLI B, THAM C K. On the design of fault-tolerant scheduling strategies using primary-backup approach for computational grids with low replication costs[J]. IEEE Transactions on Computers, 2009, 58(3): 380-393.
[15] NANDA S, CHIUEH T, BROOK S. A Survey on Virtualization Technologies[R]. USA: RPE, 2005.
[16] BALASANGAMESHWARA J, RAJU N. Performance-driven load balancing with primary-backup approach for computational grids with low communication cost and replication cost[J]. IEEE Transactions on Computers, 2013, 62(5): 990-1003.

作者简介：

王吉（1990-），男，江苏常州人，国防科学技术大学硕士生，主要研究方向为云计算、容错调度、实时系统等。

包卫东（1971-），男，内蒙古呼和浩特人，国防科学技术大学教授，主要研究方向为信息系统工程、组织分析与设计、云计算等。

朱晓敏（1979-），男，辽宁盘锦人，国防科学技术大学副教授，主要研究方向为云资源调度与优化、集群计算、多卫星调度等。