机群系统中空闲结点的功耗管理

刘勇鹏 卢 凯 迟万庆

(国防科学技术大学计算机学院 长沙 410073)

摘 要 针对机群系统中存在的大量空闲活跃结点所造成的严重能耗浪费,提出空闲结点的cache式动态功耗管理模型,即利用结点多级休眠机制,将空闲结点划分为不同休眠等级的结点集合,每级休眠状态对应一级结点储备cache,力求获得近似活跃状态的系统响应速率,以及近似最深休眠状态的能耗节省。基于cache式功耗管理模型,综合能耗与响应速率两个因素,设计了空闲结点在不同休眠状态之间的动态升降级算法、基于储备池的资源结点分配与回收算法以及储备额阈值自适应算法,以在保证系统响应速率的同时降低系统能耗。实验表明,提出的空闲结点cache式功耗管理技术在作业相对延迟仅增加0.99%的代价下,系统空闲结点功耗降低69.51%,优化效果显著。

关键词 计算机群,功耗管理,结点休眠

中图法分类号 TP315 **文献标识码** A

Power Management of Idle Nodes in Clusters

LIU Yong-peng LU Kai CHI Wan-qing

(School of Computer Science, National University of Defense Technology, Changsha 410073, China)

Abstract Existence of massive active idle nodes causes huge energy waste in large scale systems. Cache-style power management for idle nodes was proposed to schedule the power states of idle nodes. According to their different sleep states, idle nodes are placed into multiple groups with corresponding sleep states. It is expected to achieve a system response speed similar to the active state and a power saving similar to the deepest sleep state. The idle nodes are dynamically transformed between different sleep groups. Assuring response speed of system, idle node is put into a sleep state as deep as possible. In our experiments, CPMI conserves the power consumption of idle nodes by 69.51% with the cost of relative slowdown only by 0.99%.

Keywords Compute cluster, Power management, Node sleep

1 引言

随着科学应用对计算性能需求的不断提升,高性能计算系统的规模越来越庞大,计算密度越来越高,系统功耗呈摩尔定律[1]趋势飞速增长,大规模计算系统面临严峻的功耗危机。

高性能计算系统功耗惊人,最新发布的Top500中有40个系统的功耗超过一百万瓦(MW),排名前十位的系统功耗平均值高达4.34MW,其中K Computer的功耗甚至高达12.659MW[2],与一个中等城市的耗电量相当。2006年,美国数据中心的耗电总量高达610亿度,占全美电力消耗的1.5%,电费高达45亿美元[3],而且还在快速增长。由于无法满足系统供电需求,一些数据中心的建设被迫延期或取消。功耗问题严重制约着大规模计算系统的发展。

另一方面,大规模系统的应用负载通常随时间变化较大,数据中心的平均资源利用率通常只有10%~50%[4],系统中常有大量结点处于空闲状态。结点功耗与负载并不成等比关系,活跃空闲时功耗通常仍占峰值功耗的50%左右[5]。大量空闲结点的存在造成能耗的巨大浪费。

降低空闲结点功耗对大规模计算系统而言意义重大。本文面向大规模计算系统,利用结点多级休眠机制,研究空闲结点的功耗管理,旨在保证系统性能的同时,降低结点空闲时功耗,以提高系统能耗效率。

2 相关工作

动态速率调节和动态资源休眠是当前广泛支持的两大功耗管理机制。即使将空闲结点所有设备的速率都降到最低,其功耗仍比该结点的休眠功耗要高[6]。因此,利用动态休眠机制,将空闲结点置于低功耗休眠状态,在需要时再将其唤醒,是机群系统广泛采用的一种功耗管理方案[4,5,7,8]。目前,基于结点动态休眠的机群功耗管理技术主要研究负载整合,以确定合适的休眠结点数目,而对空闲结点则简单地将其关闭。在实际系统中,唤醒、休眠结点都需要额外的时间和能耗开销,而且结点通常支持多个休眠状态,不同休眠状态的待机功耗和状态转换开销也各不相同。合理选择空闲结点的休眠时机和目标休眠状态,是机群功耗管理系统需要研究的重要课题。

到稿日期:2012-06-29 返修日期:2012-10-10 本文受国家863重大项目(2012AA01A301),国家自然科学基金(60903059,61272141)资助。

刘勇鹏(1977-),男,硕士,助理研究员,主要研究方向为高性能计算、功耗管理,E-mail:liuyp@nudt.edu.cn;卢 凯(1973-),男,博士,教授,主要研究方向为高性能计算、体系结构;迟万庆(1973-),男,硕士,副研究员,主要研究方向为高性能计算、操作系统。

Gandhi 等人[9]分析了多种休眠状态对服务器功耗管理的重要性，但没有研究如何将服务器置于相应的休眠状态。Horvath 等人[10]根据系统负载变化率来预测未来性能需求，依据性能需求，考虑结点支持的多种休眠状态，求解处于各个休眠状态的最小结点数目，并将剩余结点置于最低休眠状态以实现节能。与基于负载预测来计算结点目标休眠状态不同，本文基于 cache 模型对空闲结点的状态采用分级反馈式控制，不依赖系统负载预测。Xue 等人[11]利用资源储备池思想管理空闲结点，他们在活跃待用状态储备一定数量的空闲结点，而将其余结点关机，但他们的空闲结点要么处于活跃状态，要么关机，而没有利用当前结点广泛支持的多级休眠机制。

3 空闲结点的 cache 式功耗管理模型

以是否正运行用户任务为标准，大规模计算系统中的结点可分为两大类：已分配结点和空闲结点。已分配结点是指结点已分配给用户作业，正承载作业运行；空闲结点则是指结点没有分配给任何作业，处于空转状态，不产生任何有效计算。

为降低空闲结点的能耗浪费，可将空闲结点置于低功耗休眠状态。结点通常支持多种休眠状态，例如，ACPI 定义的结点状态有 S0、S1/S2、S3、S4 和 S5，其中，S0 是活跃态，S1－S5 是不同的休眠态，休眠越深，功耗越低，进入和退出该状态的能耗开销和时间延迟也越大。

一般地，假设结点支持 M 种功耗状态，分别记为 S_0，S_1，…，S_{M-1}，其中 S_0 为活跃空闲态，S_{M-1} 为关机态。对应的待机功耗分别记为 P_0，P_1，…，P_{M-1}，对应的唤醒延迟分别记为 D_0，D_1，…，D_{M-1}，唤醒所需要的能耗开销分别记为 E_0^W，E_1^W，…，E_{M-1}^W，并且满足如下约束：

$$(\forall i,\forall j) i<j \rightarrow \begin{cases} P_i>P_j \\ D_i<D_j \\ E_i^W<E_j^W \end{cases} \quad (0\leqslant i,j\leqslant M-1) \tag{1}$$

将空闲结点休眠，旨在通过降低其空闲时功耗来实现结点空闲期间的能耗降低。因此，结点休眠时间应该足够长，以确保休眠带来的能耗降低大于状态切换自身的能耗开销。

为防止结点状态的频繁切换，并保证结点唤醒延迟在可接受范围之内，本文提出空闲结点功耗状态的 Cache 式管理模型(Cache-style Power Management for Idle nodes，CPMI)，如图 1 所示。

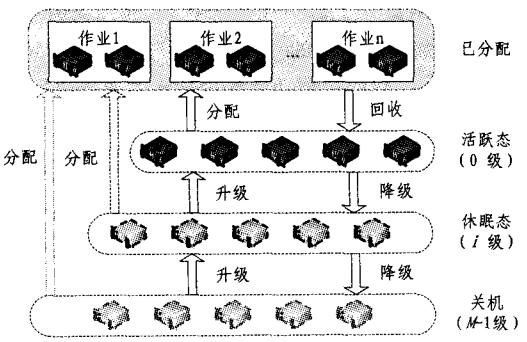

图 1 CPMI 模型

假设系统共有 N 个空闲结点，按照当前功耗状态等级分类，每一类称为一级空闲结点储备池，即处于 S_i 状态的空闲结点构成 i 级空闲结点储备池 B_i，其包含的结点数为 N_i，则有：

$$N=\sum_{i=0}^{M-1} N_i \quad (N_i\geqslant 0) \tag{2}$$

任意时刻，所有空闲结点的总功耗 P_{idle} 可以表示为：

$$P_{idle}=\sum_{i=0}^{M-1} P_i N_i \tag{3}$$

处于休眠状态的结点无法提供正常计算服务，在分配给用户使用前必须先将其唤醒。因此，结点休眠在降低系统待机能耗的同时，可能导致新来的用户负载无法立即获得足够多的活跃结点，从而影响系统的响应速率。

CPMI 为作业分配结点时，优先从高级储备池分配结点。当高一级储备池无法满足作业需求时，则将高一级储备池中所有的结点分配给作业，并继续从下一级储备池申请补足结点。假设作业申请的结点总数为 N_a，结点分配时涉及到的最低储备等级为 l，则有：

$$\sum_{i=0}^{l-1} N_i<N_a\leqslant\sum_{i=0}^{l} N_i \tag{4}$$

结合式(1)、式(4)，则可确定作业从结点分配到所有结点可用之间的延迟 D 为：

$$D=\max\{D_0,D_1,\cdots,D_l\}=D_l \tag{5}$$

为确保系统响应速率满足用户需求，各级结点储备池应储备一定数量的待用结点，以备新负载运行之需。为此，CPMI 模型为每级储备池 B_i 设置最小储备结点数阈值 R_i。CPMI 模型中结点降级到更低功耗状态时，需确保当前储备池 B_i 内的结点数不小于 R_i。

理想情况下，CPMI 为作业分配的结点都在 B_0 中命中，而大部分空闲结点则处于最低功耗状态，从而实现空闲结点 cache 式储备效果：近似最高级储备池(活跃结点)的响应速率，近似最低级储备池(关机结点)的功耗。

4 基于 CPMI 模型的功耗管理算法

CPMI 功耗管理模型中，空闲结点在不同储备池之间动态升降级，以满足节能和响应速率的综合约束。

有新作业加载时，优先从高级储备池分配结点，当作业运行完毕时，释放的结点再加入储备池。为实现节能和性能之间的有效折衷，还应根据系统运行负载实时情况，自适应调节各级储备池的储备额阈值。

相应地，本文设计了时间驱动的空闲结点功耗状态降级算法(Time-Driven Power State Degradation of idle nodes，TDPSD)、储备额驱动的空闲结点功耗状态升级算法(Reserve-Threshold Driven Power State Upgrade of idle nodes，RTDPSU)、基于 CPMI 模型的 cache 式结点分配算法(Resource Allocation with CPMI，CPMI-RA)、结点回收算法(Resource Free with CPMI，CPMI-RF)，以及 cache 失效驱动的储备额阈值自适应算法(Adaptive Threshold Adjustment according to Requirement，ATAR)。

4.1 结点升降级算法

时间驱动的空闲结点功耗状态降级算法 TDPSD 设计了时间间隔为 τ 的降级定时器，每隔 τ 时间段，定时器触发一次，执行 TDPSD 算法。为避免结点状态频繁切换震荡，本文为每级结点储备池 B_i 引入无 cache 失效持续时间阈值 T_i。当储备池的无 cache 失效持续时间超过该阈值时，申请结点降级。在结点分配过程中，一旦产生某级 cache 失效，则将应

储备池的无 cache 失效持续时间归零。

TDPSD 算法轮询所有空闲结点储备池。对于由 S_i 状态的结点构成的结点资源储备池 B_i，当 B_i 无 cache 失效持续时间 t_i 超过阈值 T_i 时，从 B_i 中选择相应的降级结点子集 DS_i，并申请将 DS_i 中的结点降低到功耗状态 S_{i+1}。TDPSD 降级时，还必须遵守资源储备额阈值 R_i 约束。当 B_i 中储备的结点数不足 R_i 时，即使 B_i 的无 cache 失效持续时间超过 T_i，也不降级。TDPSD 算法的详细描述如算法 1 所示。

算法 1 Time-Driven Power State Degradation of idle nodes (TDPSD)

```
输入:当前空闲结点储备池集合{B0,B1,…,BM-1}
输出:新空闲结点储备池集合{B0,B1,…,BM-1}
Begin
  for all node-pools Bi{
    ti=ti+τ;
    if ((ti>Ti)&&(Ni>Ri)){
      选择 Bi 的子集 DSi;
      Bi=Bi-DSi;
      Bi+1=Bi+1+DSi;
      Ni=Ni-sizeof(DSi)
      Ni+1=Ni+1+sizeof(DSi);
    }
  }
End
```

当空闲结点被分配给应用后，储备池中的结点数将相应地减少。为确保储备池中有足够多的备用结点，当储备池 B_i 中的结点数 N_i 小于 R_i 时，则从 B_{i+1} 中选择结点进行补充。备用结点的补充是一个递归过程，直到所有储备池中的结点数都满足储备额要求，如算法 2 所示。

算法 2 Reserve-Threshold Driven Power State Upgrade of idle nodes (RTDPSU)

```
输入:当前空闲结点储备池集合{B0,B1,…,BM-1}
输出:新空闲结点储备池集合{B0,B1,…,BM-1}
Begin
  for all node-pools Bi
  {
    k=i+1;
    while (Ni< Ri) {
      if (k > M)
        break;
      if (Nk>= (Ri-Ni)) {
        从 Bk 中选择(Ri-Ni)个结点到 Bi;
        Ni= Ri;
        Nk=Nk-(Ri-Ni);
      } else {
        将 Bk 中所有的结点都加入到 Bi;
        Ni= Ni+Nk;
        Nk=0;
        k=k+1;
      }
    }
  }
End
```

4.2 资源分配与回收算法

基于多级功耗状态的空闲结点储备池集合，本文设计了基于 CPMI 模型的 cache 式资源分配算法 CPMI-RA，如算法 3 所示。

算法 3 Resource Allocation with CPMI (CPMI-RA)

```
输入:当前空闲结点缓冲池集合{B0,B1,…, BM-1};
     作业申请结点数 Na
输出:新空闲结点缓冲池集合{B0,B1,…,BM-1};
     承载作业运行的结点结合 Ba
Begin
  if (Na>∑(i=0 to M-1) Ni) {
    报告资源不足;
    return;
  }
  k=0;
  n=0;
  Ba=Ø;
  while (n<Na){
    if (Nk>= (Na-n)) {
      将 Bk 中的(Na-n)个结点加入 Ba;
      Nk= Nk-(Na-n);
      n=Na;
    } else {
      将 Bk 中的所有 Nk 个结点加入 Ba;
      n=n+Nk;
      Nk=0;
      k=k+1;// 产生 k 级 cache 失效
      tk= 0; // Bk 的无 cache 失效持续时间归 0
    }
  }
  /* 调用 RTDPSU 算法,补足储备 */
  RTDPSU();
End
```

假设新加载作业申请的结点数为 N_a。CPMI-RA 首先从 B_0 中分配结点，如果 N_0 小于 N_a，则产生 0 级 cache 失效，B_0 的无 cache 失效持续时间 t_0 归零，CPMI-RA 算法除了将 B_0 中所有结点(N_0 个)分配给作业外，还从 B_1 申请(N_a-N_0)个结点；如果 N_1 仍小于(N_a-N_0)，则产生 1 级 cache 失效，B_1 的无 cache 失效持续时间 t_1 归零，除了将 B_1 中所有结点分配给该作业外，继续在 B_2 中申请($N_a-N_0-N_1$)个结点，依此类推，直到分配的结点总数满足作业需求。

储备池储备的空闲结点被分配给作业后，剩余储备结点个数减少，可能小于储备池的资源储备额阈值。因此，资源成功分配后，还需调用 RTDPSU 算法，对各级资源储备池进行补足。

当作业退出时，承载该作业运行的结点成为空闲结点，基于 CPMI 模型的资源回收算法 CPMI-RF 负载将这些结点纳入资源储备池。CPMI-RF 算法非常简单，只需将其所有结点加入到活跃状态空闲结点集合 B_0 即可，如算法 4 所示。

算法 4 Resource Free with CPMI (CPMI-RF)

```
输入:当前空闲结点缓冲池集合{B0,B1,…,BM-1};
     承载原作业运行的结点集合 Ba;结点数为 Na
输出:新空闲结点缓冲池集合{B0,B1,…,BM-1};
Begin
  B0=B0∪Ba;
  N0=N0+Na;
End
```

4.3 储备额阈值自适应算法

当新作业加载，需要为其分配相应数目的运行结点时，资源分配算法优先从最高级储备池 B_0 分配，如果 B_0 中的结点数不足，则产生 0 级 cache 失效，尝试从 B_1 中补足作业所需的结点数目，依次递推，可能产生多级 cache 失效。

产生 i 级 cache 失效，说明 B_i 当前储备的结点数 N_i 偏少。为提高 B_i 的结点储备额，提升系统响应性能，应增大其储备额阈值 R_i。为此，本文引入式(6. a)所示的根据 cache 失效动态自适应调整储备额阈值算法。其中，C_i 为此次作业分配要求 B_i 提供的结点个数，α_i 为非负系数，称为性能权重因子，用以体现系统对响应性能的敏感程度。如果 i 级储备池连续多次产生失效，则说明阈值来提高速率太慢，应相应地提升阈值来提高速率。为此，式(6)进一步引入 Tm_i 用以记录 B_i 连续发生 cache 失效的次数。

$$R_i=\begin{cases}R_i+\alpha_i\times Tm_i\times(C_i-R_i), & C_i\geqslant N_i \quad \text{(a)}\\ R_i+\beta_i\times Th_i\times(C_i-R_i), & C_i<N_i \quad \text{(b)}\end{cases} \tag{6}$$

另一方面，如果 B_i 中储备的结点在满足作业分配需求的同时，还有富余，则可将部分结点置于更低一级休眠状态，以实现节能。为此，本文相应地按照式(6. b)降低储备额阈值。其中，Th_i 为 B_i 储备结点连续富余的次数。为区别系统对节能和性能的不同需求，增加阈值动态调整的灵活性，本文引入了节能权重因子 β_i，用以反映系统的节能敏感程度。

资源分配算法在完成某个作业的结点分配后，为每级储备池调用算法 5 所示的储备额阈值自适应调节算法，根据该作业对储备池资源需求以及储备池的原储备结点数目，动态调整各级储备池的储备额阈值。

算法 5 Adaptive Threshold Adjustment according to Requirement (ATAR)

```
输入:结点储备个数 Ni;
     请求结点个数 Ci;
     结点储备额阈值 Ri
输出:结点储备额阈值 Ri
Begin
  if (Ci> Ni) {
    Tmi++;
    Thi=0;
    Ri=Ri+αi×Tmi×(Ci-Ri);
    return Ri;
  }
  if (Ci< Ni) {
    Tmi= 0;
    Thi++;
    Ri=Ri+βi×Thi×(Ci-Ri);
    return Ri;
  }
  if (Ci==Ni) {
    Tmi= 0;
    Tmi= 0;
    return Ri;
  }
End
```

5 实验测评

系统中空闲结点的时空分布与系统负载运行轨迹密切相关。只有与负载运行轨迹相结合，才能有效评估空闲结点功耗状态管理技术的实际效果。

Parallel Workloads Archive 网站发布了许多真实并行系统的运行日志，包括作业提交时间(submit time)、等待时间(wait time)、运行时间(running time)、使用处理器数目(number of allocated processors)等相关信息。本文选择其发布的系统规模最大的 ANL Intrepid[12] 数据作为本节实验的负载轨迹。ANL Intrepid 是 ANL(Argonne National Laboratory)实验室 Intrepid 系统的一段运行日志。系统包括 40960 个计算结点，每个结点包含 4 个处理器核。本文功耗管理算法从负载轨迹的 0 时刻开始运行。但是，为避免运行初期负载填充导致的实验结果偏差，本文选取该日志中 24 小时之后共 48 小时内的系统运行情况进行综合实验评估。

图 2 为本节实验中截取的 ANL Intrepid 系统运行日志的系统负载随时间变化的轨迹图，图中阴影部分表示正在运行作业的结点个数。从图中可以看出，除了第三个波峰是系统满负荷运行外，其它时间(约占 94.79%)都有结点未运行任何作业，处于空闲状态。空闲结点的大量存在，对合理管理空闲结点的功耗状态提出了现实需求。

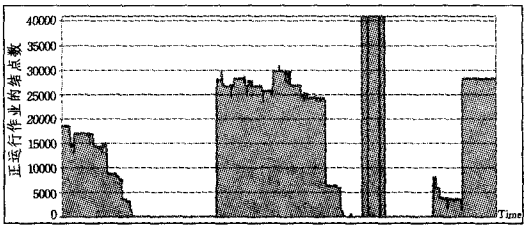

图 2 系统负载轨迹

Parallel Workloads Archive 发布的系统运行日志并不包含任何关于功耗或能耗的信息。不失一般性，本文选用典型计算结点的实测功耗作为模拟实验所需的结点功耗数据，如表 1 所列。实验过程中，空闲结点支持 3 个状态：S0、S1 和 S3。实验过程未考虑结点动态升降级时的状态切换开销。

表 1 空闲结点功耗状态表

状态	功耗(瓦)	唤醒时间(秒)
S0	207	0
S1	171	2
S3	32	10

对应空闲结点的 3 个功耗状态，实验分为 4 种情境进行对比测试：

S0。空闲结点总是处于 S0 态，即结点不进行任何休眠，实验运行轨迹与 ANL Intrepid 轨迹完全一致。

S1/S3。结点一旦空闲，立即进入对应休眠状态 S1/S3。空闲过程中，结点状态保持不变，不进行动态升降级。结点被分配给作业时，增加相应的唤醒延迟，作业启动运行时间和结束时间也相应推迟。唤醒延迟的引入对后续作业产生累积影响。在保持作业提交时间、等待时间与 ANL Intrepid 轨迹一致的前提下，为作业分配结点时要等待直到系统中有足够多的空闲结点供给该作业。唤醒过程中的功耗与休眠时功耗保持一致。

CPMI。对空闲结点的功耗状态基于 CPMI 模型进行 cache 式管理。三级储备池 B_0、B_1、B_2 对应功耗状态分别为 S0、S1 和 S3。结点分配条件与 S1/S3 情境类似，不同的是结点按照 CPMI-RA 算法取自不同的储备池。时刻 0，所有结点

处于 B_3，即 $N_0=0, N_1=0, N_2=40960, R_0=0, R_1=0$。$B_2$ 为最低功耗状态储备池，无法继续降级，因此 R_2 在实验过程中恒等于 0。TDPSD 降级算法中，$T_0=T_1=T_2=10$，选择超出 R_i 的所有结点作为降级结点集合 DS_i。ATAR 算法中各级储备池对应的 α_i 和 β_i 都设置为 1。

对应上述 4 种模拟实验情境，系统中所有空闲结点的总功耗随时间变化轨迹如图 3 所示。从图 3 可以看出，将空闲结点进入休眠状态可以显著降低系统空闲功耗。本文提出的 cache 式空闲结点状态管理技术使得系统空闲结点功耗大部分时间与所有空闲结点都进入最底层储备池（S3）的功耗相当。

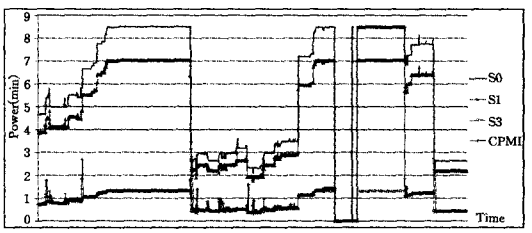

图 3　空闲结点功耗轨迹

结点从休眠状态切换到正常提供计算服务的活跃状态需要一定的时间延迟。所以，空闲结点动态休眠会对系统性能产生一定的影响。本文采用相对延迟（Relative slowdown）来衡量结点动态休眠对作业等待时间的影响。*Relative slowdown* 的计算公式如式（7）所示：

$$relative\ slowdown=\frac{wait\ time\ with\ sleep}{wait\ time} \tag{7}$$

式中，*wait time* 是无结点动态休眠情境下的作业等待时间，是作业开始运行时间与提交时间的差值；*wait time with sleep* 则是引入结点动态休眠后，作业的等待时间，也即引入结点动态休眠后作业真正开始运行的时间与作业提交时间的差值。

作业相对延迟反映了结点动态休眠对作业等待时间的影响，体现了系统反应速率。值越小，表示作业等待的相对时间越少，系统反应速率越快。

在本节实验考查的系统运行时间段内，所有作业在 4 种情境中的相对延迟如图 4 所示，图中横轴表示的作业按提交时间排序。从图 4 可以看出，相对于不进行任何休眠而言（S0），空闲结点休眠对作业相对延迟有一定程度的增加，休眠越深，相对延迟增加越多。图中 CPMI 曲线大部分与 S0 曲线（即值为 1 的直线）接近，采用 CPMI 功耗状态管理取得近似最高储备池状态的相对延迟效果。

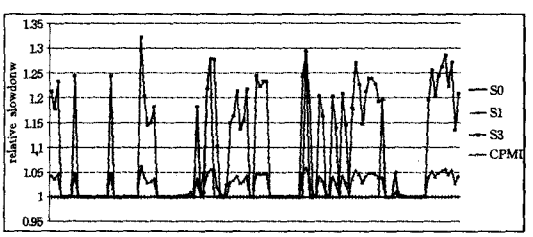

图 4　作业相对延迟

为了更准确地分析空闲结点动态休眠对功耗、性能的影响，本文将 4 个情境所有时刻的模拟值分别取代数和平均，并将所得到的平均值以无结点休眠情境（S0）为基准进行一般化，结果如图 5 所示。

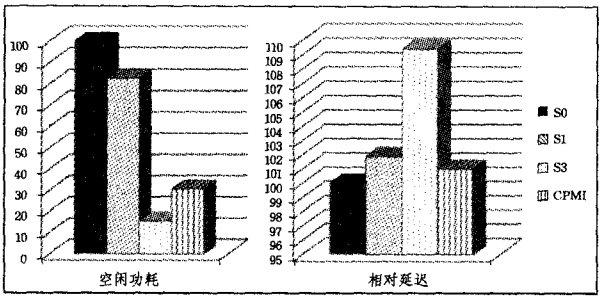

图 5　空闲结点动态休眠的结果统计

从图 5 可以看出，通过 cache 式动态休眠空闲结点，空闲结点功耗降为不进行动态休眠（S0）时的 30.49%，系统空闲功耗有了明显的降低。与此同时，系统响应速率（作业相对延迟）有所降低，但影响很小，仅有 0.99%，而且比简单地将空闲结点进入最低休眠状态（S3）所引入的 9.33%的相对延迟有明显改善。

结束语　本文针对大规模系统中空闲结点的能耗浪费，提出 cache 式空闲结点动态休眠模型，将空闲结点划分为不同休眠等级的结点储备集合，并设计了相应的结点休眠状态管理算法，力求实现系统响应速率与能耗节约之间的合理折衷，提升系统能效。

实验表明，本文提出的 cache 式空闲结点功耗状态管理技术在作业相对延迟仅增加 0.99%的代价下，系统空闲功耗降低 69.51%，优化效果显著。

空闲结点 cache 式动态休眠技术的有效性与各级资源储备池的结点储备个数密切相关。本文提出了储备额阈值的动态自适应算法，但如何调优自适应算法中的相关参数还有待进一步研究。

参 考 文 献

[1] Feng Wu-chun. Making a case for efficient supercomputing [J]. ACM Queue, 2003, 1(7): 54-64

[2] Top 500. Top500 List [OL]. http://www.top500.org, 2012

[3] U. S. Environmental Protection Agency. Report to congress on server and data center energy efficiency [OL]. http://www.energy star. gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress _Final1. pdf, 2007

[4] Krioukov A, Mohan P, Alspaugh S, et al. NapSAC: Design and implementation of a power-proportional web cluster [J]. ACM SIGCOMM Computer Communication Review, 2011, 41(1): 102-108

[5] Mustafa R M, Nishkam R, Srihari C, et al. Power management for heterogeneous clusters: an experimental study[C]// Proceedings of the 2nd International Green Computing Conference (IGCC'11). Orlando, Florida, 2011: 1-8

[6] Liu Yong-peng, Zhu Hong. A survey of the research on power management techniques for high performance systems [J]. Software Practice and Experience, 2010, 40(1): 943-964

[7] Chase J, Aderson D, Thakar P, et al. Managing energy and server resources in hosting centers[C]// Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01). Banff, Canada, 2001: 103-116

（下转第 95 页）

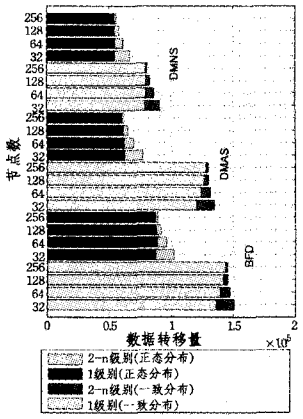

图 7 HAS 调度按级别时的数据转移量对比

从总体上看，DMNS 在数据转移量上仍然最少。也可以看出，在级别 1 产生的数据转移量比级别 $2-n$ 要大得多，且正态分布时的数据转移量比标准分布情况下要多。但在级别 $2-n$，分别在 2 种分布之间比较以及在 3 种调度之间比较，所产生的数据转移量却差不多。对于 3 种调度，交换机端口数越多，级别 $2-n$ 中产生的数据转移量越少。

7.3.2 稳定性对比

基于 DMNS、DMAS 和 BFD 这 3 种调度的 HSA 的稳定性比较如图 8 所示。可见随着交换机端口数的变化，稳定性也不同。当端口数为 64 时，3 种调度的稳定性都达到最好。总之，当端口数为 64 时基于 DMNS 的 HAS 具有最佳稳定性。

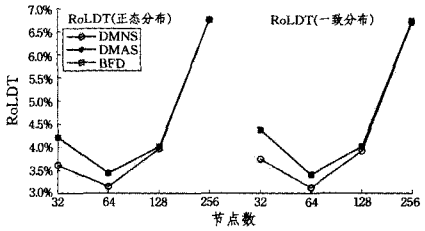

图 8 HAS 调度的 RoLDT 对比

结束语 在大型分布式计算中心的应用调度时，为了使参与计算的服务器最少和产生的数据转移量最少，我们提出了 DMNS 调度方法。在 DMNS 的基础上进一步提出的 HAS 调度算法稳定性好，时间复杂度为 $\Theta(n * \log(\log(n)))$。将 HAS 的性能与已有类似的调度算法进行仿真比较，结果显示其在稳定性和能耗方面都有占优。

在将来的调度算法研究工作中，将考虑更多节能方面的要素。例如在大型分布式计算中心每个计算节点的降温模式不相同，转移应用时，如果选择温度更低的节点，也将节约能耗和提高计算稳定性，这就成为有温度介入的调度算法研究。

参 考 文 献

[1] 林伟伟，齐德昱. 云计算资源调度研究综述[J]. 计算机科学，2012，39(10)：1-6

[2] Orgerie A-C, Lefevre L, Gelas J-P. Demystifying energy consumption in Grids and Clouds[A]// Green Computing Conference, 2010 International, 2010[C]. Chicago, IL: IEEE Conference Publications, 2010: 335-342

[3] Chase J S, Anderson D C, Thakar P N, et al. Managing energy and server resources in hosting centers[A]// Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP'01), 2001[C]. New York, NY, USA: ACM, 2001: 103-116

[4] Yao C-C. New algorithm for bin-packing[J]. Journal of ACM, 1980, 27(2): 207-227

[5] Verma A, Ahuia P, Neoqi A. Power-aware dynamic placement of HPC applications[A]// Proceedings of the 22nd annual international conference on Supercomputing (ICS'08), 2008[C]. New York, NY, USA: ACM, 2008: 175-184

[6] Johnson D. Near-Optimal Bin Packing Algorithms [M]. MIT, Cambridge, Massachusetts, 1973: 56

[7] Gambosi G, Postiglione A, Talamo M. Algorithms for the relaxed online bin-packing model[J]. SIAM Journal on Computing, 2000, 30(5): 1532-1551

[8] Sanders P, Sivadasan N, Skutella M. Online Scheduling with Bounded Migration[J]. Mathematics of Operations Research, 2009, 34(2): 481-498

[9] Srikantaiah S, Kansal A, Zhao Feng. Energy aware consolidation for cloud computing[A]// Proceedings of the 2008 conference on Power aware computing and systems (HotPower'08), 2008 [C]. Berkeley, CA, USA, USENIX Association, 2008: 10

[10] Johnson D S, Demers A, Ullman J D. et al. Worst-case Performance Bounds for Simple One-Dimensional Packing Algorithms [J]. SZAM Journal on Computing, 1974, 3(4): 299-325

[11] Baliga J, Ayre R W A, Hinton K. et al. Green Cloud Computing: Balancing Energy in Processing, Storage, and Transport[J]. Proceedings of the IEEE, 2011, 99(1): 149-167

[12] Younge A J, von Laszewski G, Wang Li-zhe, et al. Efficient resource management for Cloud computing environments[A]// Proceedings of the International Conference on Green Computing (GREENCOMP'10), 2010[C]. Washington, DC, USA, IEEE Computer Society, 2010: 357-364

(上接第 63 页)

[8] Pinheiro E, Bianchini R, Carrera E, et al. Load balancing and unbalancing for power and performance in cluster-based systems [R]. DCS-TR-440. Department of Computer Science, Rutgers University, May 2001

[9] Gandhi A, Harchol-Balter M, Kozuch M A. The case for sleep states in servers [C]// Proceedings of the 4th Workshop on Power-Aware Computing and Systems (HotPower'11). Cascais, Portugal, 2011: 6-10

[10] Horvath T, Skadron K. Multi-mode Energy Management for Multi-tier Server Clusters[C]// Proceedings of the 17th International Conference on Parallel Architecture and Compilation Techniques (PACT'08). Toronto, Canada, 2008: 270-279

[11] Xue Zheng-hua, Dong Xiao-she, Ma Si-yuan, et al. An energy-efficient management mechanism for large-scale server clusters [C]// Proceedings of the 2007 IEEE Asia-Pacific Services Computing Conference (APSCC'07). Tsukuba Science City, Japan, 2007: 509-516

[12] Feitelson D. Parallel Workloads Archive [OL]. http://www.cs.huji.ac.il/labs/parallel/workload/l_anl_int/ANL-Intrepid-2009-1.swf.gz, 2011