doi:10.11959/j.issn.1000-436x.2016008

# 温度感知的 MapReduce 节能任务调度策略

廖彬[1]，张陶[2]，于炯[3]，刘继[1]，尹路通[3]，郭刚[3]
(1. 新疆财经大学统计与信息学院，新疆 乌鲁木齐 830012；
2. 新疆医科大学医学工程技术学院，新疆 乌鲁木齐 830011；3. 新疆大学软件学院，新疆 乌鲁木齐 830008)

**摘 要：**现有的 FIFO、Fair、Capacity、LATE 及 Deadline Constraint 等 MapReduce 任务调度器的主要区别在于队列与作业选择策略的不同，而任务选择策略基本相同，都是将数据的本地性(data-locality)作为选择的主要因素，忽略了对 TaskTracker 当前温度状态的考虑。实验表明，当 TaskTracker 处于高温状态时，一方面使 CPU 利用率变高，导致节点能耗增大，任务处理速度下降，导致任务完成时间增加；另一方面，易发的宕机现象将直接导致任务的失败，推测执行(speculative execution)机制容易使运行时任务被迫中止。继而提出温度感知的节能任务调度策略，将节点 CPU 温度纳入任务调度的决策信息，以避免少数高温任务执行节点对作业整体进度的影响。实验结果表明，算法能够避免任务分配到高温节点，从而有效地缩短作业完成时间，减小作业执行能耗，提高系统稳定性。

**关键词：**绿色计算；MapReduce；任务调度；温度感知
**中图分类号：**TP393.09 **文献标识码：**A

# Temperature aware energy-efficient task scheduling strategies for mapreduce

LIAO Bin[1], ZHANG Tao[2], YU Jiong[3], LIU Ji[1], YIN Lu-tong[3], GUO Gang[3]
(1. College of Statistics and Information, Xinjiang University of Finance and Economics, Urumqi 830012, China;
2. Department of Medical Engineering and Technology, Xinjiang Medical University, Urumqi 830011, China;
3. School of Software, Xinjiang University, Urumqi 830008, China)

**Abstract:** The main difference among the existing MapReduce task chedulers such as FIFO, Fair, Capacity, LATE and Deadline Constraint is their choice of operation strat ue and job. On the count of the task selection strategies of these task schedulers are basically the same, taking the data-locality as the key factor of selection, they all ignore the current state of the temperature of the TaskTracker. The experiments show that when the TaskTracker is in a state of high temperature it will cause some negative results. On one hand, utilization of the CPU becomes higher, which means more energy is consumed at each node. And as a result of task processing speed dropping off, more time will be needed to complete the same task. On the other hand, the prone downtime phenomenon will ectly lead to the failure of the task, and speculative execution mechanism is easy to make the runtime task suspend. Temperature aware energy-efficient task scheduling strategy is put forward to solve the problem. CPU temperature of the node was put into the task scheduling decision-making information to avoid bad impact on the overall ogress of the job form the task execution nodes with a high temperature. The experimental results show that the algorithm can avoid allocating task to high temperature nodes, which effectively shorten the job completion time, reduce energy consumption of job execution and improve system stability.

**Key words:** green computing, MapReduce, task scheduling, temperature aware

**收稿日期：**2014-10-14；**修回日期：**2015-01-07
**基金项目：国家自然科学基金资助项目**(No.61562078, No.61262088, No.71261025); **新疆财经大学博士科研启动基金资助项目**(No.2015BS007)
**Foundation Items:** The National Natural Science Foundation of China (No.61562078, No.61262088, No.71261025), The Doctoral Research Foundation of Xinjiang University of Finance and Economics(No.2015BS007)

## 1 引言

大数据时代，数据从简单的处理对象开始转变为一种基础性资源，如何更好地管理和利用大数据已经成为普遍关注的话题，大数据的规模效应给数据存储、管理以及数据分析带来了极大的挑战[1]。据文献[2]统计，2007年全球数据量达到281 EB，而2007年到2011年这5年时间内，全球数据量增长了10倍。数据量的高速增长伴随而来的是存储与处理系统规模不断的扩大，这使运营成本不断的提高，其成本不仅包括硬件、机房、冷却设备等固定成本，还包括IT设备与冷却设备的电能消耗等其他开销。并且，系统的高能耗将导致过量温室气体的排放并引发环境问题。事实上，在能源价格上涨、数据中心存储规模不断扩大的今天，高能耗已逐渐成为制约云计算与大数据快速发展的一个主要瓶颈[1]。据文献[3]统计，目前IT领域的二氧化碳排放量占全球的2%，而到2020年这一比例将翻番。2008年路由器、交换机、服务器、冷却设备、数据中心等互联网设备总共消耗8 680亿度电，占全球总耗电量的5.3%。纽约时报与麦肯锡经过一年的联合调查后，最终在《纽约时报》上发表了“Power，pollution and the Internet”[4]，调查显示Goolge数据中心年耗电量约300万瓦，Facebook则达到了60万瓦，但巨大的能耗中却只有6%~12%的能耗被用于处理相应用户的请求。与此同时，Barroso等[5]对Google内部5 000多台服务器进行长达半年的调查统计结果表明，服务器在大部分时间里利用率都在10%～50%之间，服务器在负载很低(小于10%)的情况下电能消耗也超过了峰值能耗的50%。

Hadoop[6]作为新的分布式存储与计算架构，参考Google的分布式存储系统GFS[7]实现了分布式文件存储HDFS；参考MapReduce[8]计算模型实现了自己的分布式计算框架；参考BigTable实现了分布式数据库HBase。由于能够部署在通用平台上，并且具有可扩展性(scalable)、低成本(economical)、高效性(efficient)与可靠性(reliable)等优点使其在分布式计算领域得到了广泛运用，并且已逐渐成为工业与学术届事实上的海量数据并行处理标准[9]。虽然Hadoop拥有诸多优点，但是和Google服务器一样，Hadoop集群内部服务器同样存在严重的高能耗低利用率问题[10]。Hadoop主要由分布式存储系统HDFS与分布式任务执行框架MapReduce这2部分组成，现有研究大多从分布式存储系统HDFS入手解决Hadoop的能耗问题，针对MapReduce框架能耗优化方面的研究则相对较少。大量研究围绕通过对存储资源的有效调度，在不影响系统性能的前提条件下将部分存储节点调整到低能耗模式，以达到节能的目的。为提高MapReduce的能耗利用效率，本文做了如下工作。

1) 首先对MapReduce系统模型的含义进行了数学表达，并对现有的FIFO、Fair、Capacity、LATE及Deadline Constraint等MapReduce任务调度模型进行了深入的归纳分析，在此基础上总结了现有调度策略存在的问题。

2) 提出了温度感知的节能任务调度策略，将节点CPU温度纳入任务调度的决策信息，以避免少数高温任务执行节点对作业整体进度的影响。算法实现方式上提出了基于心跳信息修改及基于健康监测脚本的2种实现方案。

3) 搭建真实的实验环境，精确地测量了节点CPU温度对任务运行时间及节点能耗的影响。证明了本文算法对不同类型作业任务完成时间及作业执行能耗两方面的改进。

## 2 相关研究

传统的IT系统一方面通过超额资源供给与冗余设计以保障QoS与系统可靠性[11]，另一方面负载均衡算法专注于将用户请求平均分发给集群中的所有服务器以提高系统的可用性，这些设计原则都没有考虑到系统的能耗因素，这使IT系统的能量利用日益暴露出高能耗低效率的问题[12]。学术与工业界分别从硬件[13~15]、操作系统[16~18]、虚拟机[19~26]、数据中心[27~33]4个层次去解决IT系统的能耗问题。针对分布式计算系统的能耗问题的研究，通常以Hadoop作为研究对象，并且大多从分布式存储系统HDFS入手解决其存在的能耗问题，针对任务执行框架MapReduce能耗优化方面的研究则相对较少。

在分布式存储系统节能方面，根据软硬件角度进行划分，可分为硬件节能与软件节能两方面[34]。硬件节能主要通过低能耗高效率的硬件设备或体系结构，对现有的高能耗存储设备进行替换，从而达到节能的目的。硬件节能方法效果立竿见影，且不需要复杂的能耗管理组件；但是对于已经部署的大规模应用系统，大批量的硬件替换面临成本过高的问

题。软件节能通过对存储资源的有效调度，在不影响系统性能的前提条件下将部分存储节点调整到低能耗模式，以达到节能的目的。由于不需要对现有硬件体系进行改变，软件节能是目前云存储节能技术的研究热点。软件节能研究主要集中在基于节点管理与数据管理两方面。节点管理主要研究如何选择存储系统中的部分节点或磁盘为上层应用提供数据服务，并让其他节点进入低能耗模式以达到降低能耗的目的。节点管理中被关闭节点的选择与数据管理技术紧密相关，而目前已有的数据管理技术主要有基于静态数据放置、动态数据放置与缓存预取 3 种。其中基于静态数据放置的数据管理[35-39]根据固定的数据放置策略将数据存储到系统中各节点上后，将不再改变其存储结构。基于动态数据放置的数据管理[40~46]根据数据访问频度动态调整数据存放的位置，将访问频度高与频度低的数据迁移到不同磁盘上，对存储低频度数据的磁盘进行节能处理以降低系统能耗。基于缓存预取的数据管理[47]借鉴内存中的数据缓存思想，将磁盘中的数据取到内存或其他低能耗辅助存储设备并使原磁盘进入低能耗模式以此达到节能的目的。

研究分布式任务执行框架 MapReduce 能耗优化方面，少有的研究通过选择部分节点执行任务[48]、任务完成后关闭节点[49]、配置参数优化[50]、DVFS 调度[51]、作业调度[52]、虚拟机放置策略[53]及数据压缩[54]等方法达到提高 MapReduce 能耗利用率的目的。与 covering subset 思想相反，文献[49]提出 all-in strategy(AIS)，即将整个 MapReduce 集群作为整体用于任务的执行，当任务结束后将整个集群做节能处理(关闭节点)达到节能的目的。Chen 等[50]发现 MapReduce 框架的参数配置对 MapReduce 能耗的利用具有较大影响，通过大量的实验得到优化 MapReduce 能耗的配置参数，对提高 MapReduce 集群系统的能耗利用率具有指导意义。文献[51]中利用 DVFS(dynamic voltage and frequency scaling)技术，通过动态调整 CPU 频率以适应当前的 MapReduce 任务负载状态达到优化能耗利用的目的。文献[52]提出 Hadoop 节能适应性框架 GreenHadoop，通过合理的作业调度，在满足作业截止时间约束的前提下通过配置与当前作业量相匹配的作业处理能力(活动节点数量)，达到最小化 Hadoop 集群能耗的目的，实验证明 GreenHadoop 与 Hadoop 相比提高了 MapReduce 的能耗利用率。宋杰等[55]对云数据管理系统(包括基于 MapReduce 的系统)的能耗进行了基准测试，并定义了能耗的度量模型与能耗测试方法，证明了不同系统在能耗方面存在着较大差异，需要进一步对系统进行能耗优化。

本文与已有工作的不同在于:已有的 MapReduce 节能任务调度研究是在满足作业截止时间约束的前提下通过减小活动节点的数量，达到最小化集群能耗的目的，是从整个集群的层面进行节能，而本文则提出温度感知的任务调度模型，将节点 CPU 温度纳入任务调度的决策信息，以避免少数高温任务执行节点对作业整体进度的影响，达到缩短作业完成时间，减小作业执行能耗的目的。相比已有工作，本文是从任务调度的层面出发，实现 MapReduce 作业能耗效率的提高。

## 3　MapReduce 及其调度模型

### 3.1　MapReduce 系统模型

MapReduce 运行环境通常由多个机架 RACK 组成，而一个 RACK 内部又由多个节点服务器组成。通常情况下，MapReudce 集群由 2 个 NameNode(或 JobTracker)管理节点(主管理节点与从管理节点)与多个 DataNode(或 TaskTracker)节点构成。本文将 MapReduce 集群中所有的 DataNode 节点服务器用矩阵表示，MapReduce 集群节点矩阵如定义 1 所示。

**定义 1**　MapReduce 集群模型。设 MapReduce 集群由 $r$ 个 RACK 组成，并且设所有 RACK 中都有 $s$ 个 DataNode 节点服务器，用 $dn$ 表示 DataNode 节点服务器，将 MapReduce 集群中的所有 DataNode 节点表示为矩阵 $\boldsymbol{C}_{s\times r}$

$$\boldsymbol{C}_{s\times r}=\begin{bmatrix} dn_{11} & dn_{12} & \cdots & dn_{1r} \\ dn_{21} & dn_{22} & \cdots & dn_{2r} \\ \vdots & \vdots & & \vdots \\ dn_{s1} & dn_{s2} & \cdots & dn_{sr} \end{bmatrix} \tag{1}$$

其中，$dn_{ij}\in \boldsymbol{C}_{s\times r}\ (1\le i\le s, 1\le j\le r)$ 表示编号为 $j$ 的 RACK 中第 $i$ 个 DataNode 节点服务器。

**定义 2**　作业的任务分解模型。如图 1 所示，MapReduce 框架将作业(job)分解为多个 Map 与 Reduce 任务并行地在集群中执行，可将这个过程定义为 $f(job)\to M\cup R$，其中，$f$ 为映射函数，集合 $M$ 与 $R$ 分别表示 job 分解后的 Map 与 Reduce 任务，其任务分解模型可由式(2)表示。

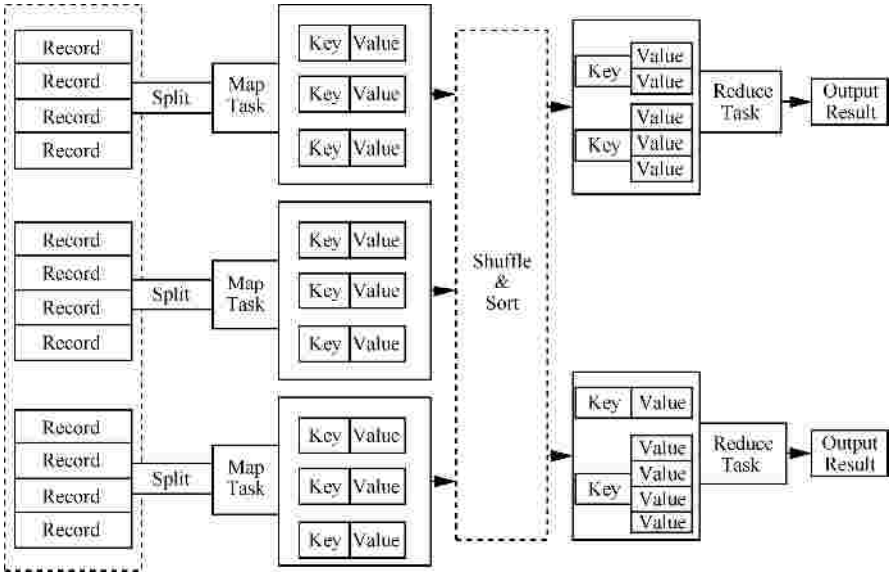



图 1 MapReduce 作业的任务分解

$$\begin{cases} f(job) \to M \cup R \\ M = \{mt_0, mt_1, \cdots, mt_{m-1}\} \\ R = \{rt_0, rt_1, \cdots, rt_{r-1}\} \end{cases} \tag{2}$$

式(2)表示作业被分解为 $m$ 个 Map 任务与 $r$ 个 Reduce 任务。其中，$M$ 表示 Map 任务集合，$mt_i \in M(0 \le i \le m-1)$ 表示任意 Map 任务；$R$ 表示 Reduce 任务集合，$rt_i \in R(0 \le i \le r-1)$ 表示任意 Reduce 任务。由于已有的 FIFO、Fair、Capacity、LATE 及 Deadline Constraint 等 MapReduce 作业调度算法已经实现作业到任务的分解。所以，作业的任务分解模型 $f(job) \to M \cup R$ 并不是本文的研究重点。作业的分解在作业的初始化阶段完成，JobTracker 根据作业输入数据量与作业配置参数(用户指定或系统默认)将作业分解成 $M$ 个 Map 任务及 $R$ 个 Reduce 任务，并将作业分解信息存储到相关数据结构中(如 XML 作业分解描述文件)，供后期调度系统执行。

**定义 3** 任务(task)资源(slot)映射模型。作业根据定义 2 被分解为 Map 与 Reduce 任务后，将由作业调度系统将任务映射到具有空闲资源槽的 DataNode 节点 $dn_{ij}$ 上执行。任务与资源映射过程可定义为映射 $f(M \cup R) \to \boldsymbol{C}_{s \times r}$，其中 $f$ 为映射函数，$M$ 与 $R$ 分别表示 Map 与 Reduce 任务的集合，$\boldsymbol{C}_{s \times r}$ 表示 MapReudce 集群中 DataNode 节点的集合。具体而言，映射模型可由式(3)描述。

$$\begin{cases} f(M \cup R) \to \boldsymbol{C}_{s \times r} \\ M = \{mt_0, mt_1, \cdots, mt_{m-1}\} \\ R = \{rt_0, rt_1, \cdots, rt_{r-1}\} \\ \boldsymbol{C}_{s \times r} = \left| dn_{ij} \right|, (1 \le i \le s, 1 \le j \le r) \end{cases} \tag{3}$$

在 MapReduce 计算模型中，任务与资源之间的映射本质上是任务调度问题。问题的核心是根据当前集群(或资源池)中各节点上的资源(如 CPU、内存、磁盘与网络等)的剩余情况与各用户作业的服务质量(QoS)要求，在资源与作业(任务)之间作出最优的匹配。由于资源剩余与用户作业 QoS 需求之间存在多样化的特点，所以 MapReduce 中的任务资源映射模型是一个多目标优化问题，属于典型的 NP 难问题。

### 3.2 MapReduce 调度模式

Hadoop 中引入资源槽(slot)的概念来抽象表示各节点(DataNode 或 TaskTracker)上的资源。Hadoop 将各节点上的资源(如 CPU、内存、磁盘与网络等)进行等量的切分，将每一份资源称作资源槽，同时将 slot 分成 Map slot 与 Reduce slot 这 2 种，对执行 Map 与 Reduce 任务时的资源使用差异进行了区分。Hadoop 将任务对多维资源的需求抽象成 slot，大大简化了资源的表示及管理问题；规定一个 task 可根据实际情况占用一个或多个 slot(大部分调度器(如 FIFO、 Fair Scheduler)只支持一个 task 占用一个 slot，而 Capacity Scheduler 可根据作业内存需求占用多个 slot)，大大简化了任务与资源之间的映射问

题。在实际运用环境中，需要根据节点硬件配置及作业特点对同节点上的 Map slot(配置参数: mapred.tasktracker.map.tasks.maximum)及 Reduce slot (配置参数：mapred.tasktracker.reduce. tasks.maximun) 数进行设置。图 2 所示为 MapReduce 作业调度模型，一个 MapReduce 作业从提交到执行的整个过程可分为 7 步。

**Step1**　用户通过调用作业提交函数将作业信息(包括作业数据及作业配置信息等)提交到 JobTracker。

**Step2**　当 JobTracker 收到用户新提交的作业后，JobTracker 将通知任务调度器 TaskScheduler 对作业进行初始化操作。

**Step3**　某个具有空闲 slot 的 TaskTracker 向 JobTracker 发送心跳(heartbeat)信息，其中包含剩余的 slot 数目，资源状态信息及能否接受新任务等信息。

**Step4**　如果某 TaskTracker 能够接受新任务，则 JobTracker 调用 TaskScheduler 中的 assignTasks 方法为该 TaskTracker 分配新的任务。

**Step5**　TaskScheduler 按照系统配置的调度策略(如 FIFO、Fair、Capacity、LATE 及 Deadline Constraint 等)为该 TaskTracker 选择出最合适的任务(或任务列表)。

**Step6**　JobTracker 将 Step5 中确定的任务(或任务列表)通过心跳应答的方式返回给 Step4 中确定的 TaskTracker。

**Step7**　当 TaskTracker 收到 JobTracker 发送的心跳信息后，如果发现心跳信息中包含需要执行的新任务，则立即启动该任务的执行。

Hadoop 中任务调度是一个可插拔的独立模块，Hadoop 集群管理员可根据自己的实际应用需求设计任务调度器，可通过参数配置项 mapred.jobtracker，taskScheduler 对调度器进行配置。任务调度器 TaskScheduler 与 JobTracker 之间存在较为密切的功能互相调用关系，JobTracker 需要调用 TaskScheduler 的 assignTasks 函数为具有空闲 slot 的 TaskTracker 分配新任务，而 JobTracker 中保存着整个集群中节点、作业及任务等相关元数据信息，而这些元数据信息是 TaskScheduler 进行调度决策时需要用到的。

### 3.3　MapReduce 能耗模型

在 3.2 节对 MapReduce 及调度模式的分析基础上，本节对 MapRedcue 能耗模型进行建模，理论上证明高温节点对能耗的影响，其中，5.2 节通过实验数据证明了高温对能耗的影响。

设某个任务 task (task $\in M \cup R$) 被映射到具有空闲资源槽(slot)的 DataNode 节点 $dn_i$ 上执行。task 任务从时间点 $t_s$ 开始，运行到时间点 $t_e$ 结束，即作业运行时间区间为 $[t_s, t_e]$。那么，运行任务 task 所需要的能耗 $E_{\text{task}}$ 可由式(4)得到。

$$E_{\text{task}} = \int_{t_s}^{t_e} P_{dni}(u_i(t))\mathrm{d}t \tag{4}$$

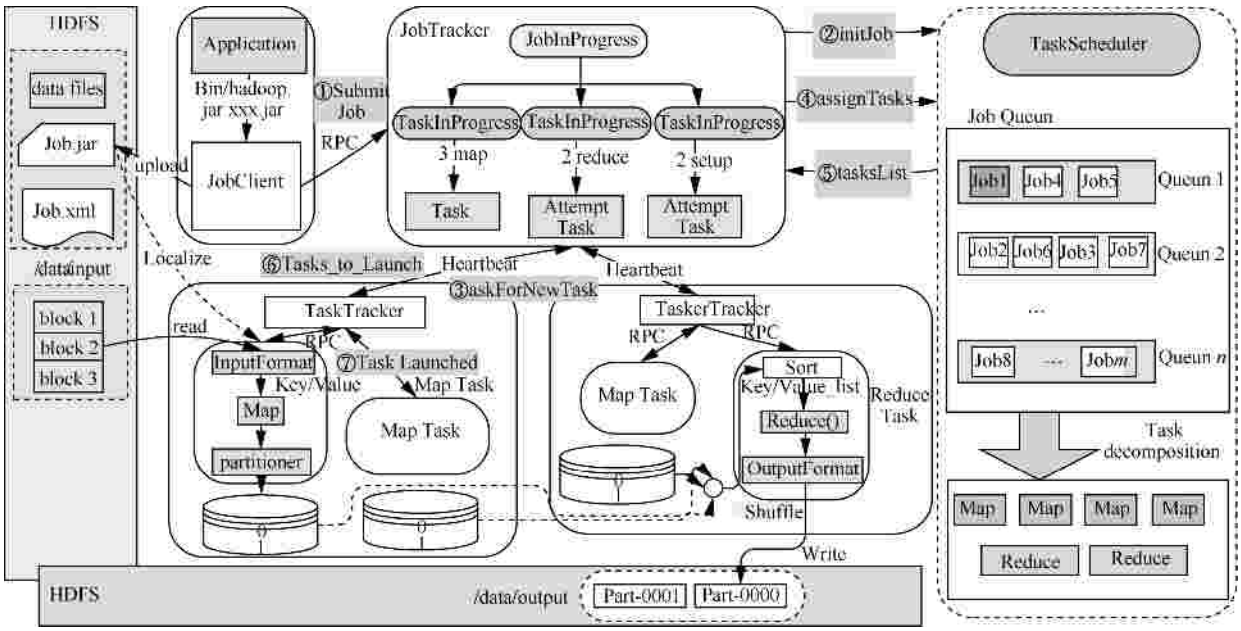



图 2　MapReduce 作业调度模型

其中，$u_i(t)$ 表示在 $t(t\in[t_s,t_e])$ 时刻节点 $dn_i$ 的系统利用率，并且存在 $u_i(t)\in[0,1]$；$P_{dn_i}(u_i(t))$ 则表示在 $t$ 时刻 $u_i(t)$ 系统利用率条件下 $dn_i$ 节点功耗。

由于 DataNode 节点功耗由静态功耗与动态功耗 2 部分组成[56]，$P_{dn_i}(u_i(t))$ 可进一步细化为 $P_{dn_i}^{\text{idle}}(u_i(t))$ 与 $P_{dn_i}^{\text{dynamic}}(u_i(t))$ 2 部分，即 $P_{dn_i}(u_i(t))$ 可表示为

$$P_{dn_i}(u_i(t))=P_{dn_i}^{\text{idle}}(u_i(t))+P_{dn_i}^{\text{dynamic}}(u_i(t)) \tag{5}$$

由于 CPU 是最主要的能耗部件，并且系统整体能耗通常与 CPU 利用率成正比[57]。同时，随着节能技术在处理器应用上的不断推广，比如 Intel 的 Speedstep 与 AMD 的 PowerNow 技术，使处理器能够根据负载动态调节性能，从而使能耗与负载之间具有较好的比例性[58]。那么，可将 $P_{dn_i}^{\text{idle}}(u_i(t))$ 与 $P_{dn_i}^{\text{dynamic}}(u_i(t))$ 分别用式(6)进行表示

$$\begin{cases}P_{dn_i}^{\text{idle}}(u_i(t))=aP_{dn_i}^{\max},\ a\in[0,1]\\ P_{dn_i}^{\text{dynamic}}(u_i(t))=(1-a)P_{dn_i}^{\max}u_i(t),\ u_i(t)\in[0,1]\end{cases} \tag{6}$$

其中，$P_{dn_i}^{\max}$ 为 DataNode 节点 $dn_i$ 满负荷(CPU 利用率 100%)时系统的功耗。$a\in[0,1]$ 为调节 $P_{dn_i}^{\text{idle}}(u_i(t))$ 与 $P_{dn_i}^{\text{dynamic}}(u_i(t))$ 之间比例的参数，其值由具体的计算机硬件体系结构决定。由式(4)~式(6)可以得到

$$E_{\text{task}}=\int_{t_s}^{t_e}(aP_{dn_i}^{\max}+(1-a)P_{dn_i}^{\max}u_i(t))\mathrm{d}t \tag{7}$$

从 MapReduce 能耗模型式(7)可以看出，计算 MapReduce 作业执行能耗的唯一前提条件是取得任务运行过程中的 CPU 利用率序列。从 5.2 节实验数据表明，过高的节点温度将加大作业运行时的 CPU 利用率。通过式(7)理论上证明了 CPU 利用率的升高将导致系统能耗的增加。

## 4 温度感知的节能任务调度算法

### 4.1 现有调度策略存在的问题

从 3.2 节中对 MapReduce 调度模式的分解可以看出，现有的 MapReduce 均为 3 级调度模型(如图 2 所示)，即当一个 TaskTracker 出现空闲资源槽时，调度器会首先选择一个作业队列，再选择作业队列中的作业，最后选择作业中的任务，并最终将选中的任务分配给这个空闲 TaskTracker。现有的 MapReduce 任务调度器的主要区别在于队列选择策略与作业选择策略的不同，而任务选择策略基本相同，都是将数据的本地性(data-locality)作为选择的主要因素。

当某 TaskTracker 出现空闲资源槽时，会立即向 JobTracker 发送心跳信息，向 JobTracker 告知自己当前的资源状态信息及能够接受新任务的意愿。而 JobTracker 也将立即通过任务调度器 TaskScheduler 为 TaskTracker 所在的空闲资源槽分配新的任务。5.2 节中实验数据表明，当节点处于高温状态时，作业完成时间变长，作业运行时功耗增加，作业运行时 CPU 利用率变高，作业(数据)处理速度变慢。实际应用环境中，节点的高温状态可能由如下 2 种原因造成。

1) 节点散热系统出现故障。节点散热系统一方面包括节点机身内部的散热系统，另一方面包括节点外部的，即数据中心(或机房)的散热系统。当节点散热系统出现故障时，节点产生的热量得不到及时的驱散，加之节点在任务运行过程中将不断产生新的热量，当热量累加到一定值时，将影响节点的运行状态，直到引起导致宕机，机器自动关闭，甚至导致硬件的烧毁。

2) 节点长时间处于高负荷运行状态。节点高负载运行时功耗达到峰值，散发出大量热量。长时间的高负荷运行容易导致节点的高温。

从上文中对已有的任务调度策略的分析可以发现调度系统并不关心拥有空闲资源槽节点当前的温度状态，一旦节点出现空闲资源槽，调度系统就尽最大努力为该节点分配合适的任务。此种调度策略最大程度上增加了系统资源利用率，使大多数作业能够更快地被处理完毕。但是，当任务被分配到处于高温状态的节点时，任务的运行会出现以下 2 种情况。

1) 任务完成时间变长。虽然 MapReduce 任务执行过程中任务之间并不是完全按照并行的方式进行的，但 Map 与 Reduce 任务之间存在不同程度的执行顺序与数据调用的制约关系(如只有当一个作业的 Map 任务成功完成的数量超过一定的阈值时，才能开始分配该作业的 Reduce 任务)，这使慢任务将造成等待时延，并最终影响到作业的整体完成时间。实际上，Hadoop 的推测执行(speculative execution)机制，容易使高温节点的运行时任务被迫中止。推测执行机制是为了防止运行速度慢的任务影响作业的整体执行速度，根据推测算法推测出“拖后腿”的任务，并为该任务启动一个备份任务，并最终选用最先成功运行完成任务的计算结果作为最终结果。推测执行机制很可能使在高温节点上执行的任务“白忙活”。

2) 任务执行失败。高温节点出现宕机将导致任务的执行失败，调度系统将对该任务重新启动。造成资源浪费的同时延迟了作业的完成时间。

基于以上考虑，4.2 节将提出温度感知的任务调度算法 2 种实现方法，将节点当前温度考虑到任务调度过程中，以此避免高温对作业执行的影响，提高系统的稳定性。

### 4.2 算法实现方法

本节主要针对 3.2 节中 MapReduce 调度模式中的 Step4 进行改进，添加任务到资源映射(或任务调度)前对节点温度的考虑，将节点当前温度考虑到任务调度过程中，以此避免高温对作业执行的影响，提高系统的稳定性。温度感知的任务调度算法有以下 2 种实现机制。

1) 将 TaskTracker 节点 CPU 温度信息添加到心跳信息类 TaskTrackerStatus 中。当 JobTracker 接受到心跳信息后，进行任务调度前判断该 TaskTracker 的 CPU 温度是否超过设定的高温阈值，如果低于高温阈值，则进行正常的调度；否则，不给该 TaskTracker 分配任务。

2) 配置 TaskTracker 健康监测脚本。节点健康监控 NodeHealthCheckerService 线程允许 Hadoop 管理员配置特定的健康监测脚本，该脚本中可添加任何检查语句作为节点是否健康运行的依据。

#### 4.2.1 基于心跳信息修改的实现方法

基于心跳信息修改的实现方法主要需要修改心跳信息类 TaskTrackerStatus 及调度器任务分配策略。

1) JobTracker 与 TaskTracker 采用基于 pull 的通信模型， JobTracker 不会主动与 TaskTracker 进行通信，而是被动等待 TaskTracker 将当前节点运行时信息(如 TaskTracker 基本信息、节点资源使用情况、各任务运行状态等)以心跳(heartbeat)的形式封装起来。这些信息都被封装到类 TaskTrackerStatus 中，该类是可序列化的，TaskTracker 每次发送心跳时，必须重新构造一个 TaskTrackerStatus 对象。

2) Hadoop 任务调度器是一个可插拔模块，用户可以自己的实际需求设计调度器，新的调度器需要继承 TaskScheduler 类。温度感知的调度器进行任务调度前需要从心跳信息中提取出 TaskTracker 温度信息，判断该 TaskTracker 的 CPU 温度是否超过设定的高温阈值，如果低于高温阈值，则进行正常的调度;否则，不给该 TaskTracker 分配任务。该算法需要在新的调度器中实现，温度感知的任务调度算法如下所示。

**算法 1　温度感知的任务调度算法**

Input:

TaskTrackerStatus　*status* /*修改后的心跳信息对象*/

TempThreshold　*threshold* /*高温阈值*/

TaskTracker　*taskTracker*　/* 待调度 TaskTracker*/

Output:

*assignTaskFlag*/*是否分配任务标识*/

Steps:

1) *nodeTemp* = *status*.getTemperature(); /*得到节点温度信息*/

2) if (*nodeTemp* ≥ *threshold*) /*如果节点实际温度大于高温阈值*/

3) set *assignTaskFlag* = false; /*不能分配新的任务*/

4) return *assignTaskFlag*; /*返回算法结果*/

5) end if

6) else if (*nodeTemp* < *threshold*) /*如果节点实际温度小于高温阈值*/

7) set *assignTaskFlag* = true; /*能够分配新的任务*/

8) revoke *TaskScheduler*.*assignTasks*(*taskTracker*); /*调用调度器 assignTasks 方法进行任务指定*/

9) return *assignTaskFlag*; /*返回算法结果*/

10) end if

算法输入参数包括心跳信息、高温阈值与待调度 TaskTracker。算法第 8)行调用调度器 TaskScheduler 类中 assignTasks 函数为待调度的 TaskTracker 分配新任务。

#### 4.2.2 基于健康监测脚本的实现方法

在 Hadoop 中，节点健康监控 NodeHealthChecker Service 线程允许 Hadoop 管理员配置特定的健康监测脚本，该脚本中可添加任何检查语句作为节点是否健康运行的依据。脚本运行时如果检测到该节点处于不健康状态，将输出以“ERROR”开头的不健康信息。NodeHealthCheckerService 线程一方面周期性调用健康监测脚本对节点进行健康检查；另一方面对脚本的输出进行检查，一旦发现脚本输出中出现“ERROR”关键字，此时便认为该节点处于不健康状态，此时该节点将被标记为“unhealthy”并通过心跳告知 JobTracker。当 JobTracker 收到节点

“unhealthy”心跳后，将该节点节点加入黑名单中，不再为其分配新的任务。当然，只要该节点上的 TaskTracker 服务处于活动状态，健康监测脚本则仍处于运行状态，当发现节点重新进入“healthy”状态后，JobTracker 会立刻将节点从黑名单中移除，使节点重新进入工作状态。脚本伪代码如下所示。

**算法 2 健康监测算法**

Input:

NodeStatus *nodestatus* /*节点状态信息*/

TempThreshold *threshold* /*高温阈值*/

Output:

*nodeState* /*节点健康状态信息*/

Steps:

1) *temperature* = *nodestatus*.getTemperature(); /*得到节点温度信息*/

2) if (*temperature* ≥ *threshold*) /*如果节点实际温度大于高温阈值*/

3) echo "ERROR, CPU temperature to high"; /*温度超过阈值，打印出错误信息*/

4) set *nodeState* = "unhealthy"; /*设置健康状态*/

5) return *nodeState*; /*返回算法结果*/

6) end if

7) else if (*temperature* < *threshold*) /*如果节点实际温度小于高温阈值*/

8) echo “OK, CPU temperature is OK”; /*温度在正常范围内，打印出信息*/

9) set *nodeState* = "healthy"; /*设置健康状态*/

10) return *nodeState*; /*返回算法结果*/

11) end if

算法第 2)行判断节点实际温度大于高温阈值，输出错误信息并设置节点健康状态为“unhealthy”；当算法判断出节点温度低于高温阈值后，输出温度正常信息，设置节点健康状态为“healthy”，算法第 10)行将节点健康状态信息返回。特别地，需要考虑怎样合理地确定高温阈值(threshold)参数的值，需要考虑集群中每台机器的硬件特点，对每个节点的高温阈值进行个性化设定。一方面要考虑到高温对作业执行的影响，另一方面也需要考虑到高温对节点硬件的损坏可能性。基于健康监测的脚步实现方法相比基于心跳信息修改的实现方法较为简单，不需要修改 MapReduce 调度源代码。

## 5 实验评价与比较

### 5.1 实验环境

项目组搭建了拥有 22 个同构节点的 Hadoop 集群实验环境，其中，NameNode 与 SecondNameNode 分别独立为一个节点，其余 20 节点为 DataNode (5RACK×4 DataNode)，实验环境拓扑结构如图 3 所示。

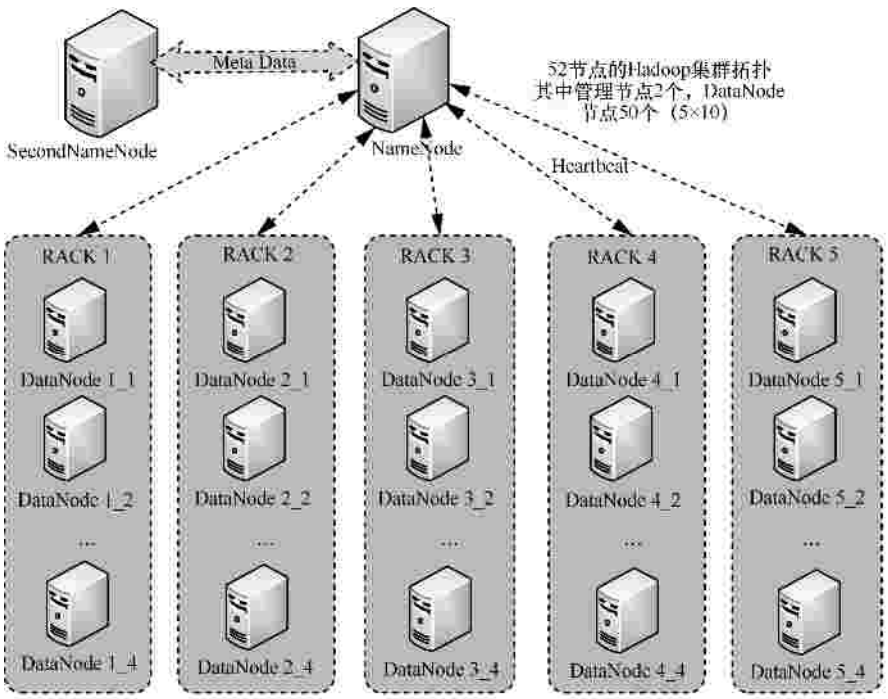



图 3 实验环境拓扑结构

为了控制实验过程中的 Map 任务数量，达到控制实验数据的统计与计算量的目的，特将数据块分块大小配置为 512 MB，即 dfs.block.size=512 MB。单个 DataNode 节点上 Map 与 Reduce 任务 Slot 资源槽数设置为 1，即配置项 mapred.tasktracker.map.tasks.maximum=1 与 mapred.tasktracker.reduce. tasks.maximum=1。能耗数据测量方面，实验采用北电电力监测仪(USB 智能版)，数据采样频率设置为 1 秒/次，各节点能耗数据(包括瞬时功率、电流值、电压值、能耗累加值等)可通过 USB 接口实时地传输到能耗数据监测机上，实现能耗数据的收集。实验总体环境描述如表 1 所示。

### 5.2　高温对任务及能耗的影响分析

为了精确地测量出节点 CPU 温度对任务运行时间及节点能耗的影响，本实验将 WordCount、TeraSort、NuthIndex、*K*-means、PageRank 这 5 种作业调度到配置相同但温度不同的节点上。本实验中涉及到的 5 种作业参数配置如表 2 所示。

按照表 3 所示配置，将作业所有任务调度到单节点上执行。实验分为 2 组，一组节点散热良好，CPU 温度控制在 50℃~75℃之间；另一组节点具有散热故障，CPU 温度控制在 75℃~90℃之间。实验进行 10 次后取平均值，分别关注 CPU 温度对作业完成时间、节点功耗、CPU 利用率以及任务的计算能力的影响。

#### 5.2.1　高温对 CPU 利用率及节点能耗的影响

如表 3 所示为高温与常温任务执行节点 CPU 利用率及功耗的对比。从表中对比数据可以看出，在执行相同作业的条件下，高温节点的 CPU 利用率都高于常温节点，并且，高温节点任务的平均功率都高于常温节点，证明了 3.3 节中提出的 MapReduce 能耗模型的正确性。

表 4 表明了高温与常温节点 CPU 利用率及能耗之间的静态特征。为了测量 CPU 利用率与系统功耗的之间动态性关系，在原 WordCount、TeraSort、NuthIndex、*K*-means、PageRank 这 5 种作业的基础上，本实验加入 Bayes 任务，并通过 10 台能耗监测仪对作业运行中的所有 DataNode 节点功耗进行实验采样，利用时间戳关联实时功耗数据与节点 CPU 利用率。

**表 1　总体实验环境描述**

| 项目 | 描述 |
|---|---|
| 操作系统 | Debian 7.0 |
| Java 版本 | 1.6 for Linux |
| Hadoop | 1.0.4 |
| 能耗数据测量 | 北电电力监测仪(USB 智能版)，标准为 GB/T17215-2003，功率误差值±0.01~0.1 W，采样频率为 1.5~3 s，单位为 kWh |
| 能耗数据采集 | 电力监测仪用电监测管理系统 V1.0.1 |
| 能耗相关单位 | 功率/W，能耗/J |
| 数据采样频率 | 1 s 采集数据 1 次 |
| 节点 CUP | Intel core2 duo E8400 3.00 GHz |
| 节点内存 | 2 GB- DDR2-800 MHz |
| 节点硬盘 | Hitachi HDP725032GLA380(320G 7 200 转/秒) |
| 网卡信息 | Realtek RTL8168/8111 PCI-E Gigabit Ethernet NIC- 100 Mbit/s |

**表 2　作业类型说明**

| 名称 | 参数配置 |
|---|---|
| WordCount | 数据总量为 976.5 MB， Map 任务数为 1， Reduce 任务数为 1 |
| TeraSort | 数据总量为 954.8 MB， Map 任务数为 1， Reduce 任务数为 1 |
| NutchIndex | Page 数量为 100 000， Map 任务数为 8， Reduce 任务数为 1 |
| *K*-means | Cluster 数为 5， Sample 数为 4 000 000， 每个 Input 文件中的 Sample 数为 4 000 000， dimensions 大小为 20， 最大迭代次数为 1， Map 任务数为 1， Reduce 任务数为 1 |
| PageRank | Page 数目为 300 000， 迭代次数设置为 3， 参数 Block 与 Block_width 分别设置为 0 与 16， Map 任务数为 1， Reduce 任务数为 1 |

表 3 高温与常温任务执行节点 CPU 利用率及功耗的对比

| 作业 | 常温 Map 阶段CPU平均利用率/% | 常温 Map 任务平均功率/W | 高温 Map 阶段CPU平均利用率/% | 高温 Map 任务平均功率/W | 常温 Reduce 阶段CPU平均利用率/% | 常温 Reduce 任务平均功率/W | 高温 Reduce 阶段CPU平均利用率/% | 高温 Reduce 任务平均功率/W |
|---|---|---|---|---|---|---|---|---|
| WordCount | 59.608 | 89.526 | 63.12 | 100.49 | 19.05 | 71.2 | 22.14 | 78.94 |
| TeraSort | 60.24 | 83.4 | 76.47 | 92.9 | 59.66 | 82.75 | 77.85 | 95.71 |
| NutchIndex | 70.54 | 89 | 72.77 | 103.5 | 65.56 | 90.5 | 68.7 | 102.2 |
| *K*-means_Cluster Iterator | 55.2 | 77.46 | 77.88 | 103.75 | 54.47 | 77.47 | 83.17 | 108.66 |
| *K*-means_Cluster Classification | 57.3 | 81.2 | 58.4 | 89.9 | N/A | N/A | N/A | N/A |
| PageRank_Stage1 | 56.24 | 83.5 | 71.17 | 102.7 | 50.4 | 80 | 77.4 | 107.5 |
| PageRank_Stage2 | 57.78 | 84 | 66.2 | 101.26 | 47.6 | 89.42 | 55.53 | 101.83 |

通过图 4 可以看出 6 种作业实时功耗与 CPU 利用率之间联系紧密；当 CPU 利用率上升时，能耗上升；当 CPU 利用率下降时，能耗下降；CPU 变化趋势与能耗变化趋势基本一致。事实上，通过图 4 表明了 3.3 节中提出的 MapReduce 能耗模型的可行性。

通过大量的能耗数据分析得出常温与高温条件下的节点功耗与 CPU 利用率之间的关系(包括理论值与实验值)如图 4 所示。其中图 4(a)表示散热良好的节点，图 4(b)表示高温节点，即具有散热故障的节点。实验值取大量测试数据的平均值，如 CPU 利用率在 50%时功耗测试数据为{83.1，85.7，88.4，87.5，89.2，84.7，90.3，83.5，82.4，87.2，82.6，85.8，86.9，85.1，84.3，88.2，86.1，83.2，88.8，86.8}，取其平均值 85.99 为其实验值。实验中本文发现散热良好(常温)的与出现散热故障(高温)的节点功耗存在较大的差异，散热良好的节点静态功耗为[64，65] W，运行时 CPU 温度稳定在 50℃~75℃；而出现散热故障的节点静态功耗为[70，72] W，运行时 CPU 在 75℃~90℃。基于 CPU 利用率估算的能耗模型的计算方法(式(5)与式(6))，可得出散热良好节点 CPU 利用率与功耗之间的理论函数为

$$P(u) = 64 + 0.5u \tag{8}$$

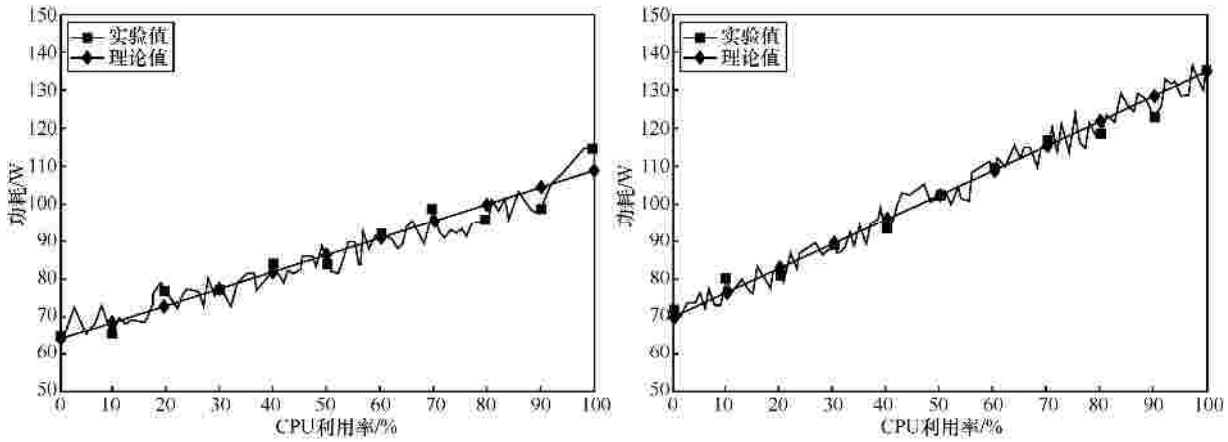
其中，$u$ 表示 CPU 利用率，$P(u)$表示 CPU 利用率为 $u$ 时节点的功耗。节点静态功耗为 64 W，峰值功耗为 110 W，由式(6)得出参数 $a \approx 0.5$。同样方法可得到出现散热故障的节点 CPU 利用率与功耗之间的理论函数

$$P(u) = 70 + 0.65u \tag{9}$$

当节点出现散热故障时，节点静态功耗为 70 W，峰值功耗为 135 W，参数 $a \approx 0.65$。式(8)、式(9)及图 4 中所示的 CPU 与功耗关系曲线与文献[57]中的结论一致，表明本文结论的正确性。

### 5.2.2 高温对任务计算能力及作业完成时间的影响

如表 5 所示为高温与常温任务执行节点任务完成时间及计算能力的对比。如表 5 数据所示，当节点处




(a) 散热良好的 Data Node 节点 (b) 散热故障的 Data Node 节点

图 4 不同温度条件下的 CPU 利用率与功耗之间的关系(理论值与实验值)

于高温状态时，5 种作业 Map 阶段与 Reduce 阶段计算能力都比常温状态时慢，表明过高的节点温度减慢任务的处理速度。高温使节点计算能力下降，会增加作业完成时间。所以，当节点处于高温状态时，5 种作业运行时间都不同程度的比常温状态时耗时长。

另外，实验过程中本文发现，当节点 CPU 温度高于 90℃并持续一段时间后，大量节点出现宕机的现象，并且温度越高，宕机的现象越严重。

### 5.3　算法对作业运行时间及能耗的影响

本实验选取了 FIFO 和 Capacity 这 2 种原作业调度策略(简称 Org-FIFO 和 Org-Capacity)，并在 FIFO 和 Capacity 作业调度策略基础上添加了温度感知的功能(简称 TempA-FIFO 和 TempA-Capacity)，在此基础上对本文提出的温度感知的任务调度模型进行实验分析。

本节实验从 22 节点构成的集群中分离出 2 组子集群进行对比实验。2 组子集群都分别拥有 10 个节点，并由 8 个散热良好和 2 个具有散热故障的节点组成(节点 CPU 温度始终高于高温阈值)。不同的是，子集群 1 利用 TempA-FIFO 及 TempA-Capacity 调度策略执行表 6 中的作业 (设置高温阈值为 80℃)，而子集群 2 则采用 Org-FIFO 及 Org-Capacity 执行表 6 中的作业。实验中采用的作业及参数配置如表 6 所示。

实验分别将表 6 中作业在 2 个子集群中运行 10 次，记录作业运行时间及能耗，取平均值后得到表 7 中的数据。其中，作业总能耗为各节点在作业运行过程中能耗的总和。实验过程中，算法设置节点 CPU 温度超过阈值 80℃时，该节点将不接受任何任务。在任务分配过程中，集群 1 中的任务全部分配到 CPU 温度低于高温阈值的节点上，高于高温阈值的节点始终处于空闲状态。而集群 2 中，任务则随机分布到集群各节点中。如表 7 所示为实验作业完成时间及能耗对比数据总表。

图 5 所示为 Org-FIFO 与 TempA-FIFO 调度下的作业完成时间对比，TempA-FIFO 较 Org-FIFO 分别提高作业 WordCount、TeraSort、NuthIndex、*K*-means、PageRank 及 Bayes 完成时间 30 s、22 s、50 s、17 s、82 s、77 s，提升率分别为 3.257%、3.509%、5.995%、3.786%、13.099%、13.775%，作业之间的差异性使各作业的提升率不同。

如图 6 所示为 Org-Capacity 与 TempA-Capacity 调度下的作业完成时间对比。Org-Capacity 较 TempA-Capacity 分别提高作业 WordCount、TeraSort、NuthIndex、*K*-means、PageRank 及 Bayes 完成时间 51 s、18 s、48 s、19 s、61 s、78 s；提升率分别为 5.849%、3.025%、5.79%、4.481%、9.967%、14.773%。

表 5　高温与常温任务执行节点任务完成时间及计算能力的对比

| 作业 | 常温 Job 运行时间/s | 高温 Job 运行时间/s | 常温 Map 任务计算能力 | 常温 Map 运行时间/s | 高温 Map 任务计算能力 | 高温 Map 运行时间/s | 常温 Reduce 任务计算能力 | 常温 Reduce 运行时间/s | 高温 Reduce 任务计算能力 | 高温 Reduce 运行时间/s |
|---|---|---|---|---|---|---|---|---|---|---|
| WordCount | 754 | 772 | 1.349 8 MB/s | 723 | 1.306 4 MB/s | 747 | 81.323 3 MB/s | 12 | 54.215 5 MB/s | 18 |
| TeraSort | 259 | 295 | 5.96 MB/s | 160 | 5.046 MB/s | 189 | 5.36 MB/s | 178 | 4.94 MB/s | 193 |
| NutchIndex | 489 | 553 | 436.68 page/s | 229 | 386.1 page/s | 259 | 233.1 page/s | 429 | 207.9 page/s | 481 |
| *K*-means_Cluster Iterator | 662 | 715 | 5 797.1 sample/s | 573 | 6 980.8 sample/s | 690 | 7 130.1 sample/s | 447 | 8 948.55 sample/s | 561 |
| *K*-means_Cluster Classification | 674 | 678 | 7 619 sample/s | 525 | 6 042 sample/s | 662 | N/A | N/A | N/A | N/A |
| PageRank_Stage1 | 246 | 277 | 3 333.333 3 page/s | 87 | 3 448.28 page/s | 90 | 1 685.393 page/s | 153 | 1 960.8 page/s | 178 |
| PageRank_Stage2 | 143 | 161 | 3 333.333 3 page/s | 90 | 3 225.8 page/s | 93 | 7 692.3 page/s | 39 | 5 882.35 page/s | 51 |

表 6　作业类型说明

| 名称 | 参数配置 |
|---|---|
| WordCount | 数据总量为 9 759.6 MB，Map 任务数为 8，Reduce 任务数为 8 |
| TeraSort | 数据总量为 9 536.7 MB，Map 任务数为 8，Reduce 任务数为 8 |
| NutchIndex | Page 数量为 1 000 000，Map 任务数为 80，Reduce 任务数为 8 |
| *K*-means | Cluster 数为 5，Sample 数为 40 000 000，每个 Input 文件中的 Sample 数为 4 000 000，dimensions 大小为 20， 最大迭代次数为 1，Map 任务数为 10， Reduce 任务数为 1 |
| Bayes | Page 数目为 50 000，分类数目为 100，参数 *ngrams*=3，Map 任务数为 10，Reduce 任务数为 1 |
| PageRank | Page 数目为 3 000 000，迭代次数设置为 3，参数 Block 与 Block_width 分别设置为 0 与 16，Map 任务数为 10，Reduce 任务数为 1 |

表 7 作业完成时间及能耗对比

| 作业名称 | TempA-FIFO 运行时间/s | Org-FIFO 运行时间/s | TempA-Capacity 运行时间/s | Org-FIFO 运行时间/s | TempA-FIFO 作业总能耗/J | Org-FIFO 作业总能耗/J | TempA-Capacity 作业总能耗/J | Org-Capacity 作业总能耗/J |
|---|---|---|---|---|---|---|---|---|
| WordCount | 891 | 921 | 821 | 872 | 675 876 | 704 566 | 612 785 | 671 478 |
| TeraSort | 605 | 627 | 577 | 595 | 491 634 | 520 982 | 465 566 | 507 383 |
| NutchIndex | 784 | 834 | 781 | 829 | 604 152 | 644 872 | 547 017 | 636 855 |
| *K*-means | 432 | 449 | 405 | 424 | 321 056 | 338 647 | 316 653 | 336 508 |
| PageRank | 544 | 626 | 551 | 612 | 424 276 | 468 362 | 404 434 | 439 103 |
| Bayes | 482 | 559 | 450 | 528 | 373 424 | 433 798 | 345 252 | 415 854 |

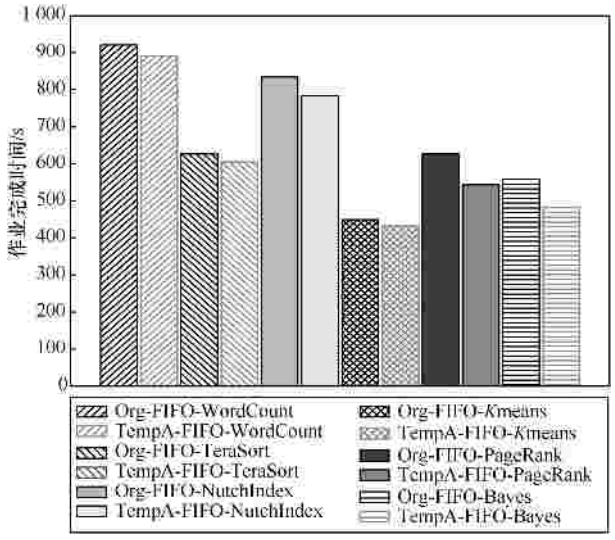



图 5 Org-FIFO 与 TempA-FIFO 作业完成时间对比

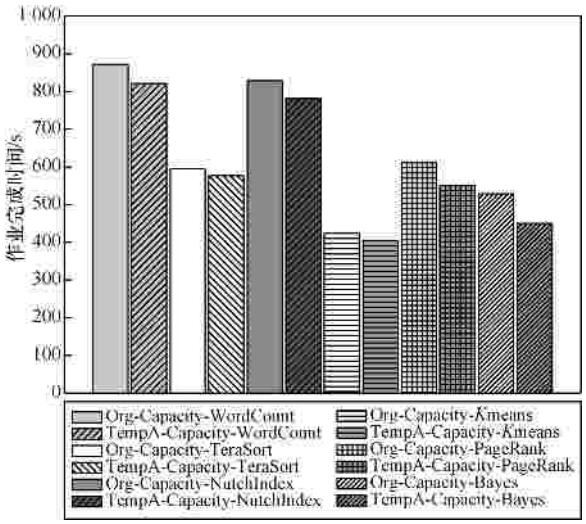



图 6 Org-Capacity 与 TempA-Capacity 作业完成时间对比

如图 7 所示为 Org-FIFO 与 TempA-FIFO 作业完成能耗对比。TempA-FIFO 相比 Org-FIFO (WordCount、TeraSort、NuthIndex、*K*-means、PageRank 及 Bayes 6 种作业)分别节能 28 690 J、29 348 J、40 720 J、17 591 J、44 086 J、60 374 J；节能率分别为 4.072%、5.633%、6.314%、5.194%、9.413%、13.918%。

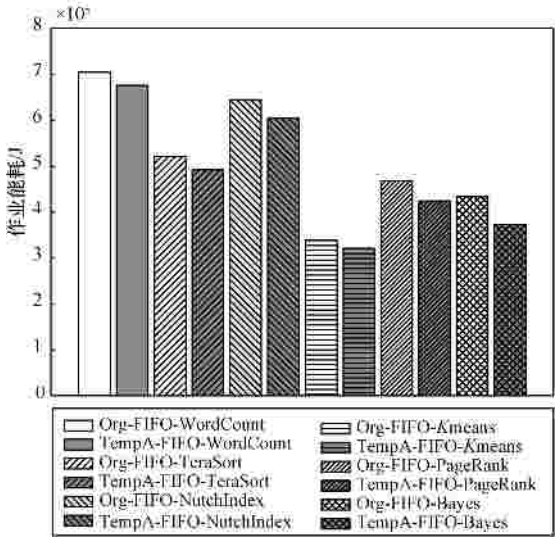



图 7 Org-FIFO 与 TempA-FIFO 作业完成能耗对比

如图 8 所示为 Org-Capacity 与 TempA-Capacity 作业完成能耗对比。TempA-Capacity 相比 Org-Capacity (WordCount、TeraSort、NuthIndex、*K*-means、PageRank 及 Bayes 6 种作业)分别节能 58 693 J、41 817 J、89 838 J、19 855 J、34 669 J、70 602 J；节能率分别为 8.741%、8.242%、14.106%、5.9%、7.895%、16.978%。

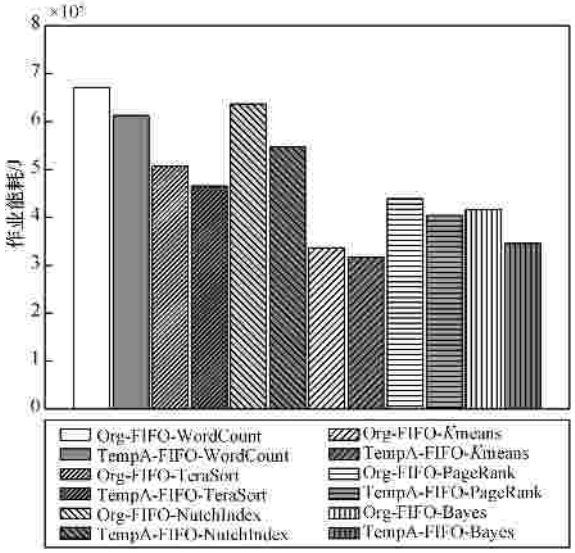



图 8 Org-Capacity 与 TempA-Capacity 作业完成能耗对比

从实验数据可以发现，本文算法对 WordCount、TeraSort、NuthIndex 及 *K*-means 4 种作业完成时间及节能效率提升并不明显，而 PageRank 及 Bayes 2 种作业有较为明显的提升。实验过程中，发现 PageRank 及 Bayes 2 种作业在运行过程中出现高温节点触发推测执行机制概率较其他 4 种作业高，由此造成作业完成时间及节能效率提升较其他 4 种作业更为明显。

## 6　结束语

大数据时代，数据量的高速增长伴随而来的是存储与处理系统规模不断的扩大，使云计算中心的能耗成本不断的提高。节能的分布式任务调度系统成为研究热点。本文通过对已有的 MapReduce 任务调度模型的深入研究，发现调度系统并不关心拥有空闲资源槽节点当前的温度状态，一旦节点出现空闲资源槽，调度系统就尽最大努力为该节点分配合适的任务。此种调度策略最大程度上增加了系统资源利用率，使大多数作业能够更快地被处理完毕。但实验表明当 TaskTracker 处于高温状态时，一方面使 CPU 利用率变高，导致节点能耗增大，任务处理速度下降，导致任务完成时间增加；另一方面，易发的宕机现象将直接导致任务的失败，推测执行机制容易使运行时任务被迫中止。所以本文提出温度感知的节能任务调度策略，将节点 CPU 温度纳入任务调度的决策信息，以避免少数高温任务执行节点对作业整体进度的影响。算法实现方式上提出了基于心跳信息修改及基于健康监测脚本的 2 种实现方案。最后，通过搭建真实的实验环境，精确的测量了节点 CPU 温度对任务运行时间及节点能耗的影响，证明了本文算法对不同类型作业任务完成时间及作业执行能耗两方面的改进。

下一步工作主要是对 MapReduce 执行能耗进行建模。MapReduce 能耗模型是将来开发 Hadoop 作业能耗监控及优化软件的理论基础，能耗模型能够实现在作业执行前对能耗进行预测，执行过程中为节能调度系统提供调度依据，执行后对作业进行能耗计算。

## 参考文献：


[1] 孟小峰，慈祥. 大数据管理：概念、技术与挑战[J]. 计算机研究与发展,2013,50(1):146-149.
MENG X F, CI X. Big data management: concepts, techniques and challenges[J]. Journal of Computer Research and Development, 2013, 50(1): 146-149.

[2] GANTZ J, CHUTE C, MANFREDIZ A, et al. The diverse and exploding digital universe: an updated forecast of worldwide information growth through[EB/OL]. http://www.ifap.ru/library/ book268.pdf.

[3] Global action plan, an inefficient truth[R/OL]. Global action plan report, 2007. http://globalactionplan.org.uk.

[4] TIMES N Y. Power, pollution and the Internet [EB/OL]. http://www.nytimes.com/2012/09/23/technology/ data-ceneters-waste-vast-amounts-of-energy-belying-industry-image.html.

[5] BARROSO L A, HLZLE U. The datacenter as a computer: an introduction to the design of warehouse-scale machines [R]. Morgan: Synthesis Lectures on Computer Architecture, Morgan & Cla l Publishers, 2009.

[6] BORTHAKU D. The hadoop distributed file system: architecture and design[J]. Hadoop Project Website, 2007, 11(11):1-10.

[7] GHEMAWAT S, GOBIOFF H, LEUNG S T. The google gile system[C]//19th ACM Symposium on Operating System Principles. New York, ACM, c2003: 29-43.

[8] DEAN J, GHEMAWAT S. MapReduce: simplifed data processing on large clusters[C]//The Conference on Operating System Design and Implementation (OSDI). New York, ACM, c2004: 137-150.

[9] 王鹏，孟丹，詹剑锋，等. 数据密集型计算编程模型研究进展[J]. 计算机研究与发展,2010,47(11):1993-2002.
WANG P, MENG D, ZHAN JF, et al. Review of programming models for data-Intensive computing[J]. Journal of Computer Research a Development, 2010, 47(11): 1993-2002.

[10] LI D, WANG J E. Energy efficient redundant and inexpensive disk array[C]//The ACM SIGOPS European Workshop. New York, ACM, c2004: 29-35.

[11] 林闯，田源，姚敏. 绿色网络和绿色评价：节能机制、模型和评价[J].计算机学报,2011,34(4):593-612.
LIN C, TIAN Y, YAO M. Green network and green evaluation: Mechanism, modeling and evaluation [J]. Chinese Journal f Computers, 2011, 34(4): 593-612.

[12] 廖彬，于炯，张陶，等. 基于分布式文件系统 HDFS 的节能算法[J]. 计算机学报, 2013, 36(5):1047-1064.
LIAO B, YU J, ZHANG T, et al. Energy-efficient algorithms for distributed file system HDFS[J]. Chinese Journal of Computers, 2013, 36(5): 1047-1064.

[13] ALBERS S. Energy-efficient algorithms [J]. Communications of the ACM, 2010, 53(5): 86-96.

[14] WIERMAN A, ANDREW L L, TANG A. Power-aware speed scaling in processor sharing systems[C]//The 28th Conference on Computer Communications (INFOCOM 2009). Piscataway, NJ, c2009: 2007-2015.

[15] ANDREW L L, LIN M, WIERMAN A. Optimality, fairness, and robustness in speed scaling designs[C]//ACM International Conference on Measurement and Modeling of International Computer Systems (SIGMETRICS 2010). New York, ACM, c2010: 37-48.

[16] NEUGEBAUER R, MCAULEY D. Energy is just another resource: energy accounting and energy pricing in the nemesis OS[C]//The 8th IEEE Workshop on Hot Topics in Operating Systems. Piscataway, NJ, c2001: 59-64.

[17] FLINN J, SATYANARAYANAN M. Managing battery lifetime w h energy-aware adaptation [J]. ACM Transactions on Computer Systems (TOCS), 2004, 22(2): 179-182.

[18] MEISNER D, GOLD B T, WENISCH T F. PowerNap: eliminatin server idle power [J]. ACM SIGPLAN Notices, 2009, 44(3): 205-216.

[19] YE K, JIANG X, YE D, et al. Two optimization mechanisms to improve the isolation property of server consolidation i virtualized multi-core server[C]//The 12th IEEE International Conference on High Performance Computing and Communications. Melbourne, Australia, c2010: 281-288.

[20] CHOI J, GOVINDAN S, JEONG J, et al. Power consumption prediction and power-aware packing in consolidated environments[J]. IEEE Transactions on Computers, c2010, 59(12):1640-1654.

[21] YE K, JIANG X, HUANG D, et al. Live migration of multiple virtual machines with resource reservation in cloud computing environments[C]//The 4th IEEE International Conference on Cloud Computing. Washington, USA, c2011: 267-274.

[22] LIAO X, JIN H, LIU H. Towards a green cluster through ynamic remapping of virtual machines[J]. Future Generation Co ter Systems, 2012, 28(2):469-477.

[23] JANG J W, JEON M, KIM H S, et al. Energy reduction in consolidated servers through memory-aware virtual machine scheduling[J]. IEEE Transactions on Computers, 2011, 99(1): 552-564.

[24] WANG X, WANG Y. Coordinating power control and performance management for virtualized server cluster[J]. IEEE Tra ions on Parallel and Distributed Systems, 2011, 22(2):245-259.

[25] WANG Y, WNAG X, CHEN M, et al. Partic: power-aware response time control for virtualized web servers[J]. IEEE Transactions on Parallel and Distributed Systems, 2011, 22(2):323-336.

[26] DASGUPTA G, SHARMA A, VERMA A, et al. Workload management for power efficiency in virtualized data-centers[J]. Communications of the ACM, 2011, 54(7): 131-141.

[27] SRIKANTAIAH S, KANSAL A, ZHAO F. Energy aware consolidation for cloud computing [J]. Cluster Computing, 2009, 12(1): 1-15.

[28] GARG S K, YEO C S, ANANDASIVAM A, et al. Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers [J]. Journal of Parallel and Distributed Computing, 2010, 71(6): 732-749.

[29] KUSIC D, KEPHART J O, HANSON J E, et al. Power and performance management of virtualized computing environments ia lookahead control [J]. Cluster Computing, 2009, 12(1): 1-15.

[30] SONG Y, WANG H, LI Y, et al. Multi-tiered on-demand resource scheduling for VM-based data center[C]//Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing nd the Grid (CCGrid 2009). Piscataway, NJ, c2009: 148-155.

[31] GMACH D, ROLIA J, CHERKASOVA L, et al. resource pool management: Reactive versus proactive or let's be friends [J]. Computer Networks, 2009, 53(17): 2905-2922.

[32] BUYYA R, BELOGLAZOV A, ABAWAJY J. Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges[J]. Eprint Arxiv, 2010,12(4): 6-17.

[33] KIM K H, BELOGLAZOV A, BUYYA R. Power-aware provisioning of cloud resources for real-time services[C]//The 7th International Workshop on Middleware for Grids. New York, c2009: 1-6.

[34] **王意洁, 孙伟东, 周松, 等. 云计算环境下分布存储关键技术**[J].**软件学报**,2012,23(4):962-986.
WANG Y J, SUN W D, ZHOU S, et al. Key technologies of distributed storage for cloud computing[J]. Journal of Software, 2012,23(4): 962-986.

[35] GREENAN K M, LONG D D E, MILLER E L, et al. A spin-up saved is energy earned: achieving power-efficient, erasure-coded storage[C]//The HotDep 2008. Berkeley: USENIX Association, c2008:4.

[36] WEDDLE C, OLDHAM M, QIAN J, et al. A gear-shifting power-aware raid[J]. ACM Transactions on Storage, 2007, 3(3): 1553-1569.

[37] LI D, WANG J. Conserving energy in conventional disk based RAID systems[C]//The 3rd Int Workshop on Storage Network Architecture and Parallel I/Os (SNAPI 2005). Piscataway, NJ, c2005: 65-72.

[38] YAO X, WANG J. Rimac: a novel redundancy-based hierarchical cache architecture for energy efficient, high performa storage systems[C]//The EuroSys. New York, ACM, c2006: 249-262.

[39] PINHEIRO E, BIANCHINI R, DUBNICKI C. Exploiting redundancy to conserve energy in storage systems[C]//The SIGMetrics Performance 2006. New York, ACM, c2006: 15-26.

[40] NARAYANAN D, DONNELLY A, ROWSTRON A. Write off-loading: practical power management for enterprise storage[J]. ACM Transactions on Storage (TOS), 2008, 4(3):253-267.

[41] STORER M, GREENAN K, MILLER E, et al. Replacing tape with energy efficient, reliable, disk-based archival storage[C]//The FAST 2008. New York, ACM, c2008: 1-16.

[42] ZHU Q, CHEN Z, TAN L, et al. Hibernator: Helping disk arrays sleep through the winter[C]//The 20th ACM Symposium on Operating Systems Principles (SOSP). New York, ACM, c2005: 177-190.

[43] VASIC N, BARISITS M, SALZGEBER V. Making cluster applications energy-aware[C]//The ACDC 2009. New York, ACM, c2009: 37-42.

[44] **廖彬, 于炯, 孙华, 等. 基于存储结构重配置的分布式存储系统节能算法**[J].**计算机研究与发展**, 2013, 50(1): 3-18.
LIAO B, YU J, SUN H, et al. Energy-efficient algorithms for distributed storage system based on data storage structure reconfiguration[J]. Journal of Computer Research and Development, 2013, 50(1): 3-18.

[45] LIAO B, YU J, SUN H, et al. A QoS-aware dynamic data replica deletion strategy for distributed storage systems under cloud computing environments[C]//Cloud and Green Computing (CGC), 2012 Second International Conference. IEEE, c2012: 219-225.

[46] **廖彬, 于炯, 钱育蓉, 等. 基于可用性度量的分布式文件系统节点失效恢复算法**[J].**计算机科学**, 2013, 40(1):144-149.
LIAO B, YU J, QIAN Y R, et al. The node failure recovery algorithm for distributed file system based on measurement of data availability[J].Computer Sicence, 2013, 40(1): 144-149.

[47] ZHU Q, DAVID F M, DEVARAJ C F, et al. Reducing energy consumption of disk storage using power-aware cache management[C]//The HPCA 2004. Piscataway, NJ, c2004: 118-129.

[48] LEVERICH J, KOZYRAKIS C. On the energy (in)efficiency f hadoop clusters [J]. ACM SIGOPS Operating Systems Review, 2010, 44 (1): 61-65.

[49] LANG W, PATEL J M. Energy management for mapreduce clusters[J]. Proceedings of the VLDB Endowment, 2010, 3(1-2): 129-139.

[50] CHEN Y, KEYS L, KATZ R H. Towards energy effcient mapreduce [R]. Berkeley: EECS Department, University of California, 2009-10-9.

[51] WIRTZ T, GE R. Improving MapReduce energy efficiency for computation intensive workloads[C]//Green Computing Conference and Workshops (IGCC). IEEE, c2011: 1-8.

[52] GOIRI Í, LE K, NGUYEN T D, et al. GreenHadoop: leveraging green energy in data-processing frameworks[C]//The 7th ACM european conference on Computer Systems. ACM, c2012: 57-70.

[53] CARDOSA M, SINGH A, PUCHA H, et al. Exploiting Spatio-Temporal Tradeoffs for Energy Efficient MapReduce in t Cloud[D]. Department of Computer Science and Engineering, University of Minnesota, 2010.

[54] CHEN Y, GANAPATHI A, KATZ R H. To compress or not to compress-compute vs. IO tradeoffs for mapreduce energy efficiency[C]//The First ACM SIGCOMM Workshop on Green Networking. New Delhi, India, c2010: 23-28.

[55] **宋杰, 李甜甜, 朱志良, 等. 云数据管理系统能耗基准测试与分析**[J]. **计算机学报**,2013,36(7):1485-1499.
SONG J, LI T T, ZHU Z L, et al. Benchmarking and analyzing the energy consumption of cloud data management system [J]. Chinese Journal of Computers, 2013, 36(7): 1485-1492.

[56] ANDREWS M, ANTA A F, ZHANG L, et al. Routing for energy minimization in the speed scaling model[C]//The 29th IEEE Conference on Computer Communications (INFOCOM'10). San Diego, USA, c2010: 1-9.

[57] BARROSO L A, HOLZLE U. The case for energy-proportional computing[J]. Computer, 2007, 40(12): 33-37.

[58] **林闯,田源,姚敏.绿色网络和绿色评价:节能机制、模型和评价**[J].**计算机学报**,2011,34(4):593-612.
LIN C, TIAN Y, YAO M. Green network and green evaluation: Mechanism, modeling and evaluation [J]. Chinese Journal of Computers, 2011, 34(4): 593-612.

## 作者简介：

**廖彬（1986-），男，四川内江人，博士，新疆财经大学副教授，主要研究方向为绿色计算、数据库系统理论及数据挖掘等。**

**张陶（1986-），女，新疆乌鲁木齐人，新疆医科大学讲师，主要研究方向为云计算、医学大数据分析等。**

**于炯（1964-），男，北京人，博士，新疆大学教授、博士生导师，主要研究方向为网格计算、大数据与云计算等。**

**刘继（1974-），男，新疆乌鲁木齐人，博士，新疆财经大学教授，主要研究方向为信息管理及数据挖掘。**

**尹路通（1992-），男，河南信阳人，新疆大学硕士生，主要研究方向为节能计算、大数据计算模式等。**

**郭刚（1990-），男，新疆乌鲁木齐人，新疆大学硕士生，主要研究方向为节能计算、大数据计算模式等。**