基于镜像的在线文件系统检查工具的研究与实现①

王雅楠[1], 陈香兰[1], 代 栋[1], 孙明明[1], 周学海[1,2]

[1](中国科学技术大学 计算机科学与技术学院, 合肥 230026)
[2](中国科学技术大学 苏州研究院, 苏州 215123)

摘 要: 设计和实现了一种全新的在线文件系统检查工具—OnlineFSCK(On-Line File System Checker). OnlineFSCK 可以对在线的文件系统进行一致性检查. 利用 OnlineFSCK 对文件系统进行检查时, 文件系统可以继续正常提供服务. 提出和实现了一种对在线文件系统生成镜像的算法, 并将这个算法与现有的文件系统检查工具相结合, 最终达到对在线文件系统进行检查的目的. 针对 ext3 文件系统实现了 OnlineFSCK 的原型, 实验结果表明, OnlineFSCK 在满足对性能的要求的前提下, 能够达到与传统文件系统检查工具相同的检查能力. OnlineFSCK 的实现中没有修改文件系统的内核源码, 可以扩展支持多种文件系统.

关键词: 文件系统检查; 在线检查; 镜像; 文件系统一致性; 在线镜像

Research and Implementation of an Online File System Checker Based on Image

WANG Ya-Nan[1], CHEN Xiang-Lan[1], DAI Dong[1], SUN Ming-Ming[1], ZHOU Xue-Hai[1,2]

[1](School of Computer Science and Technology, University of Technology and Science of China, Hefei 230026, China)
[2](Suzhou Institute for Advanced Study, University of Technology and Science of China, Hefei 230026, China)

Abstract: This paper designs and implements OnlineFSCK (On-Line File System Checker), which checks the file system while it is online without modifying the in-kernel file system codes. The file system can continue providing service while it is being checked by OnlineFSCK. This paper proposes and implements an runtime imaging algorithm which, combined with the existing file system checkers, can provide us the ability to check the consistency of the existing running file systems. The prototype is implemented for ext3 file system. Experiments shows that our OnlineFSCK solution can guarantee the correctness as traditional file system checkers while keeping the performance of the production system neglectable impacted. As there is no need to modify the in-kernel codes to support it, Online FSCK can be easily extended to support other file systems.

Key words: file system checker; online checker; file system image; file system consistency; online filesystem imaging

基于磁盘的文件系统在存储体系中占据着最重要的中心地位. 文件系统是由许多部分组成的一个极其复杂的系统. 尽管在文件和存储方面, 已经有很多人做了大量优秀的工作, 但是文件系统错误仍然时有发生. 存储栈中的不同的部分都有可能成为造成文件系统错误的来源, 比如: 磁盘介质, 机械部件, 驱动器固件, 数据传输层, 总线控制器, 驱动程序, 以及文件系统本身代码中存在的 bug 等[1-3].

文件系统中的有些错误可以被检查报告, 甚至可以被纠正. 但是还有些错误, 他们在发生时没有任何迹象, 威斯康辛的研究人员把这类错误称为“静默错误”[4]. 研究表明, 这种“静默错误”会在系统中传播, 甚至导致整个系统不可用. 当(我们怀疑)文件系统中存在错误时, 传统的做法是运行文件系统检查工具对其进行检查, 如 fsck[5]. fsck 能够检查和修复文件系统中的错误, 使其重新回到一致性的状态. 但是 fsck 要求其检查的文件系统是静态的, 因此在用 fsck 对文件系统进行检查之前, 必须把文件系统卸载. 由于文件

① 基金项目:江苏省产学研前瞻性联合研究项目(BY2009128)
收稿时间:2013-01-10;收到修改稿时间:2013-03-08

系统的检查是IO密集型的任务, 因此fsck的执行是一个非常慢的过程[6]. 在一个要求 7x24 小时服务的生产系统中, 这样长时间的文件系统离线是不可接受的. 然而, 目前仍然没有一个能够对已有的文件系统进行在线检查的工具出现.

文件系统检查工具要求被检查的文件系统是静态的, 然而一个在线的文件系统中的元数据是不断的变化的, 这正是在线文件系统检查的难点所在. 因此, 在线文件系统检查的关键问题就是, 如何在保证文件系统服务可用性的前提下, 获取到在线文件系统的一个稳定状态的视图来作为文件系统检查工具的输入. 在本文中, 我们设计和实现了一个在线文件系统检查工具——OnlineFSCK. 在 OnlineFSCK 的实现中没有任何对文件系统内核代码的修改, 因此它具有较好的兼容性, 能够用于对已有的运行中的文件系统(目前针对 ext3 实现)进行在线检查. 在 OnlineFSCK 的实现中, 本文的基本思路是改变检查的直接对象. 我们设计了一个高效的在线文件系统镜像算法, 然后通过对文件系统镜像的检查, 达到检查物理文件系统目的. 由于 OnlineFSCK 没有修改任何内核代码, 因此我们相信它可以很容易的移植到 ext3 之外的其他文件系统中去. OnlineFSCK 的创新性在于以下几点:

① 在线检查. OnlineFSCK 能够对在线的文件系统进行检查, 这使得系统管理员能够随时检查以获知文件系统的健康状况;

② 高性能. OnlineFSCK 的运行给系统带来的性能下降是可控的, 并且能够保持在很低的水平;

③ 通用性. OnlineFSCK 不需要修改内核代码, 能够直接应用于已有的甚至正在运行的文件系统, 并且可以简单的扩展支持多种文件系统.

在本文的第 1 部分介绍了 OnlineFSCK 的体系结构. 第 2 部分将详细描述 OnlineFSCK 在 ext3 文件系统上的实现. 在第 3 部分中, 我们将展示和分析实验结果. 第 4 部分和第 5 部分分别是对相关工作的讨论和总结.

1 OnlineFSCK的体系结构

OnlineFSCK 的基本思想是先对在线的文件系统创建镜像, 通过对镜像的检查达到检查文件系统的目的. OnlineFSCK 能够保证在生成镜像和对镜像执行检查的过程中, 物理文件系统能够继续提供读写服务, 并且对系统性能的影响能够控制在较低的水平. 图 1 展示了 OnlineFSCK 的总体框架. OnlineFSCK 由内核驱动模块(Kernel Driver, 简称 KD)和用户态控制程序(Usermode Controllor, 简称 UC)两部分组成. KD 是一个可加载的内核模块, 负责完成过滤写操作数据流中的元数据、记录这些元数据的块号以及把元数据从物理文件系统复制到镜像文件等工作. UC 是一个用户态进程, 主要负责的工作是生成初始的不完整镜像并在 KD 的支持下, 逐步修复镜像, 生成与物理文件系统一致的元数据镜像, 以及对镜像进行检查.

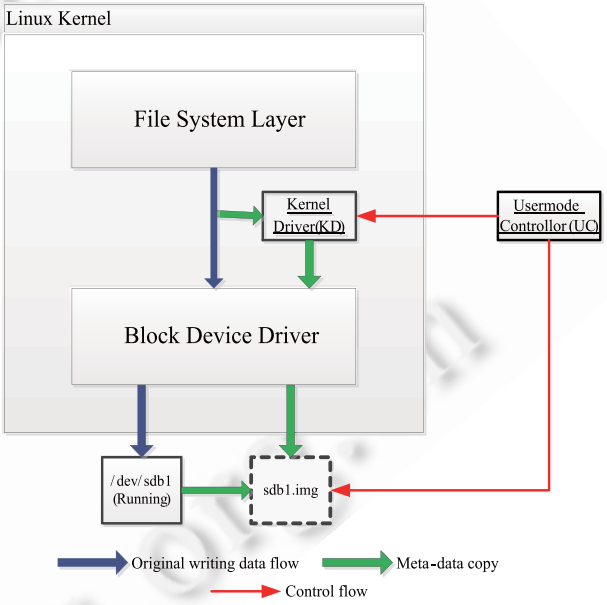

图 1 OnlineFSCK 的体系结构

在用 fsck 对离线文件系统进行检查时, fsck 可以修复检查到的文件系统错误. 但是, 由于在线的文件系统中的元数据一直处于变动中, 因此在不修改内核代码以提供文件系统级支持的情况下, 对在线的文件系统进行修复是非常困难的. 为了保持 OnlineFSCK 的通用性, 我们选择了折中的方案. OnlineFSCK 会对文件系统做 fsck 所做的全部检查工作, 而不去做在线的修复. 之所以这样做, 是基于两点考虑: 首先, 通常情况下, 文件系统中存在错误的概率较小; 其次, 如果检查出文件系统中存在错误, 那么把文件系统卸载之后再花时间去修复(一般为几分钟到几十分钟)是值得的, 因为这样可以防止错误进一步的传播, 而导致更严重的问题[4].

2　OnlineFSCK的实现

前文述及, OnlineFSCK 中, 被直接检查的对象是文件系统的镜像, 因此实现的关键就在于如何生成一个与在线的物理文件系统一致的镜像, 并且对系统性能的影响较小. 在 OnlineFSCK 中, 我们使用和 e2fsprog 中相同格式的镜像. 文件系统的镜像, 是包含文件系统全部元数据的稀疏文件. ext3 文件系统的元数据包括超级块、块组描述符、块位图、索引节点位图、索引节点表和间接块. 在线文件系统中的这些元数据都在被使用, 随时会改变. 由于文件系统检查工具在检查时也要读取这些数据, 这时就会存在冲突, 因此我们不能用传统的文件系统检查工具对在线文件系统进行检查[7]. 这也是无法对在线的文件系统进行修复的原因. 如果直接从在线文件系统中读取元数据来生成镜像, 那么由于元数据时刻在改变, 这样生成的镜像与物理文件系统的元数据是不一致的, 无法正确反映物理文件系统的健康状态. 为了解决这个问题, OnlineFSCK 利用了文件系统冻结的操作. 我们设计了一个迭代收敛的在线镜像算法. 这个算法能够使 OnlineFSCK 所导致的持续冻结时间可控(比如小于 1 秒), 使其适用于实际的生产系统. 另外, 我们还使用了其他一些技巧使 OnlineFSCK 对系统性能的影响保持在一个生产系统能够接受的水平.

2.1 OnlineFSCK 的处理流程

图 2 展示了用 OnlineFSCK 进行一次在线检查的处理流程. 如图 2 中的 A 部分, OnlineFSCK 一次完整的运行包括两个阶段:

① 第一阶段, 生成初始镜像

在这个阶段, UC 直接扫描物理文件系统, 将其元数据复制到镜像文件(该文件位于其他文件系统中). 图2.B展示了这个阶段的流程. 在这个过程中, 文件系统照常运行, 提供读写服务. 与此同时, KD 持续过滤物理文件系统的写操作数据流, 将其中的元数据块号记录到 w_map 中. w_map 是我们设计的“双位图”机制的其中一个位图, 在第二阶段将描述“双位图”机制.

② 第二阶段, 迭代生成最终镜像并检查

在这个阶段中, OnlineFSCK 通过一个迭代收敛的过程, 逐步修正镜像, 最终生成一个与物理文件系统一致的镜像. 我们设计了“双位图”的机制来记录元数据, 他们分别是“w_map”和“r_map”. 每个位图都对应于整个待检查文件系统, 其中的每个位对应于一个块

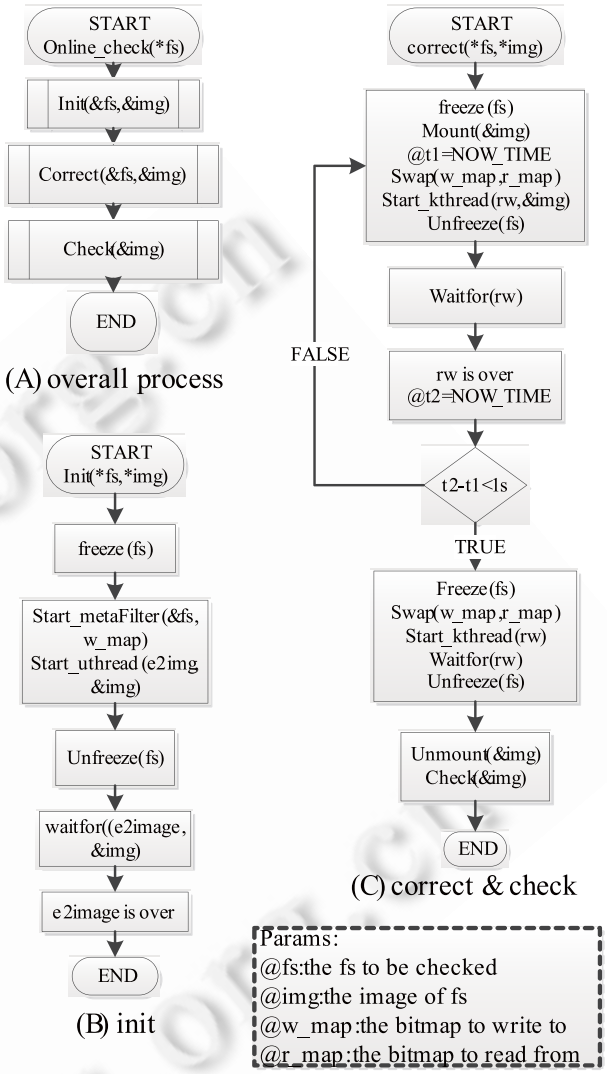

图 2　OnlineFSCK 的处理流程

(block). w_map 是写位图(writing map), 是指在当前迭代步中, 这个位图是用来写入的, 即将过滤到的元数据的块号标记到这个位图中去. r_map 是读位图(reading map), 是指在当前迭代步中, 这个位图是用来从其中读取元数据(块号)的. 在每个迭代步的开始时刻, OnlineFSCK 冻结文件系统, 然后切换 w_map 和 r_map 并立即将文件系统解冻. 因此, r_map 实际上是记录着上一个迭代步中所过滤到的写操作所修改的元数据的块号. 在每个迭代步中, OnlineFSCK 利用一个内核线程“rw”把 r_map 中记录的元数据块从物理文件系统复制到镜像文件中去. 与此同时, KD 持续过滤文件系统的写操作流, 并把其中的元数据块号记录到 w_map 中. 当 r_map 中记录的块全部被复制到镜像文

件中时, 一个迭代步就到达终点. 如果这个迭代步所消耗的时间大于 1 秒(或其他设定的可接受的持续冻结时间值, 下同), 则继续开始下一个迭代步. 否则, 我们认为此时 w_map 中仅有极少的几个块被记录(因为这个迭代步经过的时间很短), 那么我们就进入最后一个迭代步. 在最后一个迭代步中, 我们冻结文件系统, 启动 rw 线程把 w_map 中记录到的块号复制到镜像文件中. 当 rw 线程执行完毕的时候, 将文件系统解冻. 此时的镜像文件就是最终版本的镜像文件, 其中包含物理文件系统的所有元数据. 现在我们对这个镜像文件执行文件系统检查. 对这个镜像文件进行一致性检查, 就等价于对物理文件系统进行检查. 图2.C展示第二阶段的流程.

利用上述算法可以高效的对在线的文件系统进行镜像. OnlineFSCK 对文件系统的持续冻结时间和对系统性能带来的影响都能保持在较小的水平, 完全能够达到生产环境的要求.

2.2 算法证明

接下来我们从理论上说明以上算法能够使文件系统冻结时间收敛到我们所期望的水平.

假设文件系统的负载基本稳定, 把物理文件系统的元数据复制到镜像文件的速率是 R. 文件系统元数据的总大小是 S_0, 第 1 次迭代(这里指生成初始镜像阶段)将消耗的时间为 T_0, 则有:

$$T_0 = \frac{S_0}{R} \tag{1}$$

假设在 T_0 时间内被修改的元数据量为 S_1, 在 T_0 时间内被修改的总数据量的最大值为 W_1, 第 2 次迭代所消耗的时间是 T_1, 则有:

$$T_1 = \frac{S_1}{R}, S_0 = W_1 > S_1 \tag{2}$$

以此类推, 假设在第 n-1 次迭代中文件系统被修改的脏元数据量为 S_n, 被修改的总数据量的最大值为 W_n, 第 n 次迭代所消耗的时间为 T_n, 则有:

$$T_n = \frac{S_n}{R}, S_n = W_n > S_n \tag{3}$$

根据(1), (2), (3)可得:

$$S_0 > S_1 > S_2 > \mathrm{L} > S_n$$

由以上各式可得:

$$T_0 * R > T_1 * R > T_2 * R > \mathrm{L} > T_n * R \Rightarrow T_0 > T_1 > T_2 > \mathrm{L} > T_n$$

由上式可知, OnlineFSCK 的算法是收敛的. 当经过若干次迭代之后, 在一个迭代步中复制元数据所耗费的时间总能控制在小于某个值. OnlineFSCK 的实现中, 这个值设置为 1 秒, 也可根据不同需求进行配置.

3 测试结果及性能评价

我们从正确性和性能两方面对传统文件系统检查工具 e2fsck 和我们的 OnlineFSCK 进行了测试. 测试环境的配置如表 1 所示.

表 1 测试环境配置

配置	参数
CPU	Intel Core2 2.66Hz
内存	2G
硬盘	WDC WD1600AAJS
操作系统发行版本	Suse Server 11
Linux 内核版本	2.6.32-x86_64

3.1 准确性测试和评价

我们用 debugfs 进行故障注入, 并比较 OnlineFSCK 和 e2fsck 对文件系统错误的检查结果. 我们构造了超级块、组描述符、索引节点和目录结构等多个方面的文件系统错误. 对每种错误, 我们先用 e2fsck 进行检查, 并且不进行修复, 然后再将文件系统挂载, 并用 OnlineFSCK 进行检查. 检查结果显示 OnlineFSCK 和 e2fsck 都能检查出我们构造的所有文件系统错误. 由于我们采用的是和 e2image 相同的镜像文件格式, 因此这个结果是我们预料之中的. 但是在测试中我们还发现, OnlineFSCK 会额外报一些“孤儿节点”的错误. 这是由于一种使用文件的特殊方法造成的: 在 linux 中, 我们可以用 open 系统调用打开一个文件, 然后对其进行 unlink 操作后继续使用这个打开的文件, 直到用 close 系统调用把这个文件关闭. 在这个文件被 unlink 之后和 close 之前, 这个文件的 inode 就会被加入到系统中的“孤儿节点链表”中. 因此, 在线的文件系统中存在孤儿节点是正常的, 不能以此判断文件系统存在错误. 在生产环境中, 为了避免这种“误报”, 我们也可以简单的修改对镜像进行检查的代码, 使其忽略这种检查.

3.2 性能测试和评价

时间效率是衡量文件系统检查工具的最重要标准. 我们从两个方面来评价 OnlineFSCK 的时间效率. 一方面, 一次检查所耗费的总时间直接反映了文件系统检查工具的性能. 另一方面, 我们认为对于一个在线

文件系统检查工具来说, 运行过程中的文件系统冻结时间是最重要的衡量标准. 因此, 我们从运行时间和冻结时间两个方面来考量 OnlineFSCK 的性能. 考虑到文件系统大小和文件系统使用率都会影响在线文件系统的运行情况, 我们构造如表 2 所示的 4 种用例对 OnlineFSCK 进行测试. 实验结果表明, OnlineFSCK 的运行时间保持在能够接受的范围内, 并且文件系统的持续冻结时间保持在 1 秒以下, 完全适用于生产环境.

表 2　性能测试用例

	文件系统容量	使用率（4K 文件填充）
case1	10G	0%
case2		80%
case3	30G	0%
case4		80%

为使测试环境更加接近生产环境, 我们利用 stress 进行压力测试. 在用 OnlineFSCK 对在线文件系统进行检查的过程中, 我们在该文件系统中运行 stress 来模拟生产环境的系统负载.

图 3 中的数据显示, 随着文件系统容量和文件系统使用率的增长, 两者运行时间增长趋势基本一致. OnlineFSCK 比 fsck 运行一次所需的时间略长, 这个结果是在我们预料之中的. 这是由于在线文件系统在被检查的同时, 还要承受正常读写服务的负载. 由于 OnlineFSCK 在检查的同时, 文件系统仍能继续提供读写服务, 因此我们认为检查时间的增加是可以接受的.

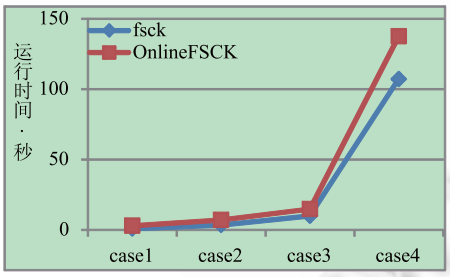

图 3　运行时间(秒)比较

如 2.1 节中所述, OnlineFSCK 的运行过程中需要短暂的冻结文件系统以获取一致点. 图 4 中显示的是持续冻结时间的测试结果. 从图 4 中数据可以看出, OnlineFSCK 所导致的持续冻结时间非常短. 当文件系统容量增大或者文件系统使用率增加时, 累计冻结时间和持续冻结时间都会随之增加. 但是由于 OnlineFSCK 的算法能够在一定程度上自动调节迭代次数, 保证持续冻结时间小于 1 秒, 这保证了 OnlineFSCK 能够满足实际应用的需求.

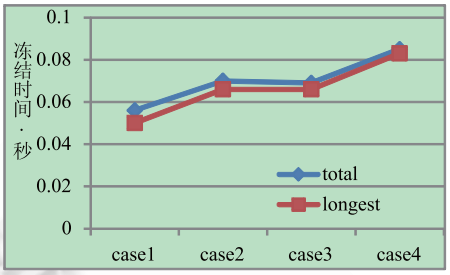

图 4　冻结时间(秒)

4　相关工作

基于磁盘的文件系统已被作为主流存储设备使用数十年, 文件系统检查工具仍然在蓬勃的发展中. 近年来, 在文件系统可靠性方面的工作主要可以分为两种思路: 一种是构造一个外部的工具对文件系统的一致性进行检查, 另外一种是试图使文件系统自身对每个操作的一致性进行验证.

绝大多数的文件系统都依赖于类似 fsck 的文件系统检查工具来进行一致性的检查. 然而这种传统的文件系统检查工具要求文件系统离线才能保证检查结果的正确性. 由于文件系统检查是一个非常耗时的过程[6], 一次文件系统检查可能导致长时间的文件系统离线, 这在实际生产环境中是难以忍受的. 文[6]中提出了一种减少 fsck 运行时间的方法. SQCK[8]是一个利用类似 SQL 的指令语言进行优化和改进的离线文件系统检查工具. 尽管这些方法能改善传统文件系统检查工具的性能, 但是他们仍然还是要求文件系统离线才能进行检查. McKusick 提出一种后台运行 fsck 的方法[9]. 这种方法能够在不卸载文件系统的情况下用 fsck 对其进行一致性检查. 但是这种方法可能导致长时间的文件系统冻结. 文献[7]提出了一种在线文件系统检查方法. 这种方法要求通过修改文件系统的内核代码, 在文件系统中植入一个摘要数据库, 以支持检查工具. 因此, 这种方法不适用于对已有的文件系统进行在线检查.

Recon[10]是一个运行时检查的框架. 它利用一些不变量在文件系统内部进行检查和修复. 文献[11]提出用模型检查来发现文件系统中存在的错误. 这些方法都要求对文件系统的代码进行大量的修改, 因此也都不能够用于已有的文件系统的在线检查.

5 结论及未来工作

本文提出了一种新型的文件系统检查工具OnlineFSCK, 详细的阐述了OnlineFSCK的原理和实现细节, 并针对ext3实现了OnlineFSCK的原型. 实验结果表明, OnlineFSCK 能够在满足对性能的要求的前提下对在线的文件系统进行检查, 并得到与传统检查工具一致的检查结果. OnlineFSCK 对文件系统的持续冻结时间保持在一个很小的水平, 满足了实际生产环境服务器对性能的要求. 另外, OnlineFSCK的实现中不需要对内核代码进行任何修改. 下一步工作, 我们将着手把OnlineFSCK扩展到ext3以外的其他文件系统.

参考文献

1 Zhang Y, Rajimwale A, Arpaci-Dusseau AC, Arpaci-Dusseau RH. End-to-end data integrity for file systems: a ZFS case study. Proc. of the 8th USENIX conference on File and storage technologies. USENIX Association, 2010: 3–16.

2 Ghemawat S, Gobioff H, Leung ST. The Google file system. (ACM Special Interest Group on Operating Systems)SIGOPS Operating Systems Review. ACM,2003,37(5):29–43.

3 Vijayan P, Bairavasundaram LN, Agrawal N, Gunawi HS, Arpaci-Dusseau AC, Arpaci-Dusseau RH. IRON File Systems. (ACM Symposium on Operating Systems Principles) SOSP'05. Brighton, 2005: 206–220.

4 Bairavasundaram LN, Goodson GR, Schroeder B, Arpaci-Dusseau AC, Arpaci-Dussea RH. An analysis of data corruption in the storage stack. Proc. of the 6th USENIX Conference on File and Storage Technologies. San Jose, 2008: 1–16.

5 McKusick MK, Kowalski TJ. Fsck−The UNIX†File System Check Program. Unix System Manager's Manual-4.3BSD Virtual VAX-11 Version.1986.

6 Henson V, Brown Z, Ts'o T, van de Ven. Reducing fsck time for ext2 file systems. Linux Symposium. Ottawa, 2006: 395–407.

7 Gunawi H. Improving File System Reliability and Availability with Continuous Checker and Repair, 2011.

8 Gunawi HS, Rajimwale A, Arpaci-Dusseau AC, Arpaci-Dusseau RH. SQCK: a declarative file system checker. Proc. of the 8th Symposium on Operating Systems Design and Implementation(OSDI'08). USENIX Association, 2008: 131–146.

9 McKusick MK. Running fsck in the background. Usenix BSDCon 2002 Conference Proc. USENIX Association, 2002: 55–64.

10 Fryer D, Sun K, Mahmood R, Cheng TH, Benjamin S, Goel A, Brown AD. Recon: verifying file system consistency at runtime. Proc. of the 10th USENIX Conference on File and Storage Technologies. USENIX Association, 2012: 73–86.

11 Yang J, Twohey P, Engler D, Musuvathi M. Using model checking to find serious file system errors. ACM Transactions on Computer Systems, 2006,24(4):393–423.

(上接第 213 页)

明显. 对于实际的电子地图而言, 因节点数目较多, 车辆在寻找到最优路径所节省下的时间可以弥补算法搜索所浪费的时间, 试验结果证明, 该算法在路径规划寻找全局最优解上具有一定的优势.

参考文献

1 梁栋,史忠科.半结构化道路下智能车辆路径规划与控制算法研究.交通运输系统工程与信息,2011,11(2):44–51.

2 张广林,胡小梅,柴剑飞,赵磊,俞涛.路径规划算法及其应用综述.现代机械,2011,5:85–89.

3 丁世飞,苏春阳.基于遗传算法的优化 BP 神经网络算法与应用研究.Proc. of the 29th Chinese Control Conference, 2010,7:2425–2428.

4 乔维德.遗传算法和神经网络在交通事故预测中的应用.电气传动自动化,2008,30(1):41–44.

5 飞思科技产品研发中心.神经网络理论与 MATLAB7 实现.北京:电子工业出版社,2005.99–108.

6 吴琦,胡德金,张永宏,徐鸿钧.网络建模在套料钻性能预测中的应用.兵工学报,2006,3(27):494–497.

7 刘志伟.遗传进化型神经网络体系结构研究[硕士学位论文].合肥:合肥工业大学,2007.

8 田明星.路径规划在车辆导航系统中的应用研究.北京:北京交通大学,2009.

9 宋立成.智能交通动态路径诱导算法的研究.济南:山东大学,2008.

10 李慧,杨东梅,沈洁,高凯.BP 神经网络在路径规的中的应用.应用科技,2004,31(9):15–16.