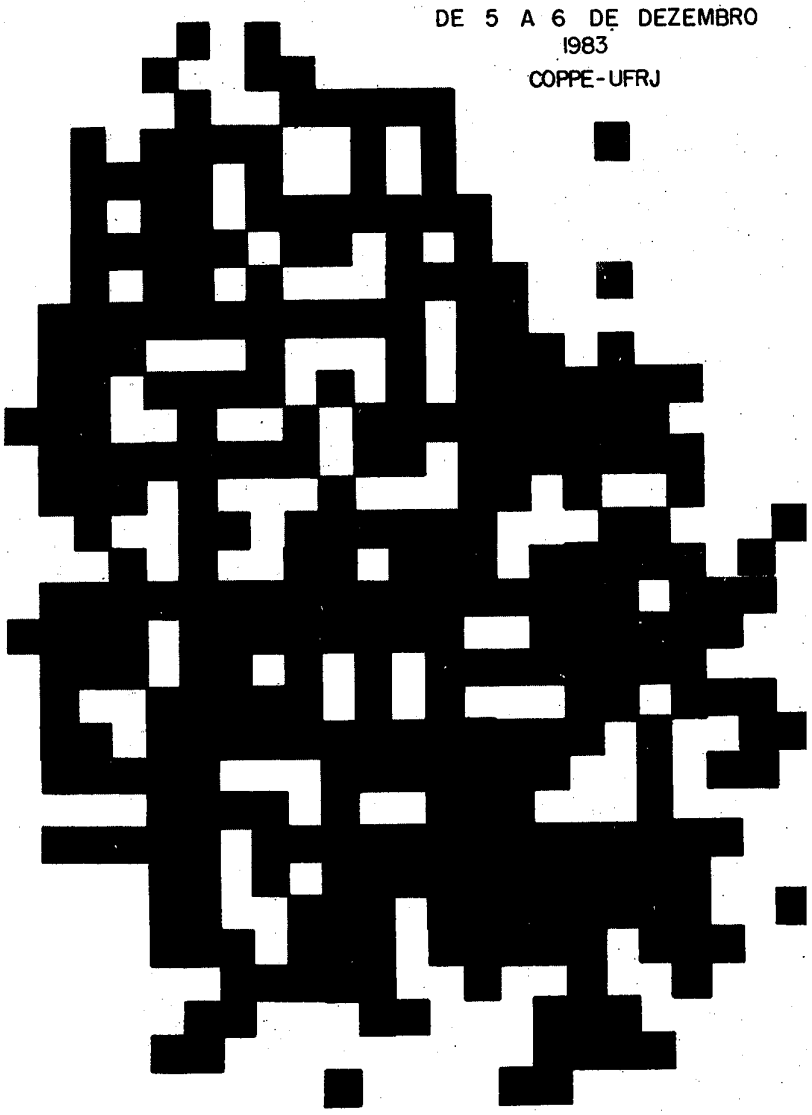
III SIMPÓSIO

SOBRE DESENVOLVIMENTO DE SOFTWARE BÁSICO PARA MICROS

DE 5 A 6 DE DEZEMBRO
1983
COPPE-UFRJ



ANAIS

A N A I S

3° SIMPÓSIO SOBRE
DESENVOLVIMENTO DE SOFTWARE BÁSICO
PARA MICROS

RIO DE JANEIRO, 5 E 6 DE DEZEMBRO DE 1983

SBC/SOCIEDADE BRASILEIRA DE COMPUTAÇÃO
- COMISSÃO ESPECIAL PARA LINGUAGENS E SISTEMAS DE PROGRAMAÇÃO

COPPE-UFRJ/COORDENAÇÃO DE PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

DI-PUC-RJ/DEPARTAMENTO DE INFORMÁTICA DA PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

CNPq/CONSELHO NACIONAL DE DESENVOLVIMENTO CIENTÍFICO E TECNOLÓGICO

COMISSÃO ORGANIZADORA:

Michael A. Stanton, Coordenador (DI-PUC-RJ)
Roberto da Silva Bigonha (UFMG)
Sueli Mendes dos Santos (COPPE-UFRJ)

Implementação de um Método de Acesso Indexado com utilização de árvores B+.

por: Ewerton Aluisio Pinto Vieira e
Mário André Guimarães.

Departamento de Informática - PUC/RJ

Resumo

O trabalho descreve o projeto e implementação de um Método de Acesso Indexado (MAI) constituído de um conjunto de primitivas para manipulação de registros de tamanho variável contendo chaves únicas também de tamanho variável. O ambiente destino foi o sistema operacional CP/M nas versões 8 e 16 (CP/M-86) bits e a linguagem utilizada, C.

Estruturalmente, o método pode ser dividido em dois níveis independentes (a menos do tamanho das páginas lógicas). O primeiro chamado Método de Acesso Paginado (MAP) é o responsavel pelo mapeamento em memória secundária das páginas lógicas, gerenciando a alocação de novas páginas e a liberação de páginas descartadas promovendo sua reutilização quando necessário. O algorítmo de alocação de áreas em memória principal é do tipo LRU minimizando o número de acessos a memória secundária. O segundo nível é construído sobre o primeiro e é o que efetivamente implementa as primitivas de acesso indexado. A estrutura de armazenamento escolhida, árvores B+, minimiza o número de acessos na pesquisa direta a um registro e permite também uma eficiente pesquisa sequencial nos registros que ficam naturalmente em ordem crescente. O tamanho variável de registros e chaves permite também uma maior eficiência no que diz respeito à utilização de espaço e, portanto, na altura da árvore de acesso.

Palavras Chaves: Arquivo indexado , Arvores B, CP/M, Linguagem C, Método de Acesso.

I) INTRODUÇÃO

O trabalho surgiu da necessidade de criar um método de acesso que permitisse trabalhar com um grande volume de dados. Tratando-se de sistemas interativos, o tempo de resposta deveria ser baixo. Outra característica comum às nossas aplicações era a necessidade de trabalhar com acessos direto e sequencial às informações. Depois de um levantamento bibliográfico (4,5,6,7) e orientados pelo professor Daniel A. Menascé, optamos individualmente, por um método de acesso utilizando árvores B+. Embora estivéssemos visando aplicações distintas, resolvemos

trabalhar juntos com o propósito de criar um conjunto de subrotinas básicas dentro de princípios de um projeto estruturado (generalidade, transportabilidade, modularidade, confiabilidade, eficiência e simplicidade) ao invés de criar dois sistemas isolados.

Utilizou-se a estrutura conhecida como árvore B+ na implementação. Descrevemos um pouco de suas características abaixo.

No final da década de 60, várias indústrias de computadores e grupos de pesquisa desenvolveram sistemas de arquivos e seus respectivos metodos de acesso. Um dos sistemas que mais prevaleceu foi o de Arvores B proposta por Bayer e McCreight na Boeing Scientific Research Labs (9). Esse método consiste num mecanismo interno de indexação de baixo custo para a maioria das operações.

O grande problema da árvore B refere-se à pesquisa sequencial. Para resolver este problema foi desenvolvido uma variação da árvore B, denominada por autores como B+ (8) ou B* onde os níveis mais altos das árvores são apenas índices, enquanto que as folhas possuem as chaves e as informações associadas. As folhas constituem uma lista encadeada, permitindo um eficiente acesso sequencial.

Vários Sistemas de Gerência de Banco de Dados utilizam árvores B+ ou alguma variação da mesma (11,12). Entre eles, podemos citar:

- Sistema R (BD relacional p/ maq.grande porte)
- INGRES (BD relacional p/ maq. médio porte)
- O IMS através do VSAM (método de acesso p/ o BD hierárquico p/ maq. de grande porte)
- dBASEII (BD relacional p/ microcomputador)
- MDBSIII (BD de redes p/ microcomputador)

Discute-se no trabalho cada um dos dois níveis da implementação, as opções e decisões tomadas, restrições e dificuldades encontradas ao longo do desenvolvimento, além das possíveis alterações e melhoramentos que poderiam ser realizados.

II) ARQUITETURA

O ambiente onde os sistemas seriam inicialmente utilizados consistiria de um microcomputador baseado no microprocessador 8086 (1), construído no LESC (Laboratório de Engenharia e Sistemas de Computação) da PUC/RJ. Este computador está configurado com 128k bytes de memória e dois acionadores de discos flexíveis de oito polegadas, densidade simples, face simples. O sistema operacional disponível e o CP/M-86 (2) que oferece algumas primitivas de acesso a disco bastante limitadas. Também disponíveis para o CPM-86 encontram-se o ASM86, montador de linguagem de máquina e a linguagem de alto nível C (3). Foi esta última a escolhida para a implementação efetiva do Método de Acesso Indexado devido às suas interessantes características de transportabilidade, estruturação, modularidade e eficiência.

Seria produzida então uma biblioteca de funções primitivas que estenderiam a linguagem C para que se pudesse

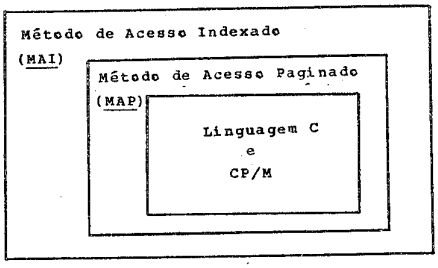
manipular arquivos indexados com o mínimo de dificuldade possível. Além disso, seria interessante que o método de acesso fosse o mais independente possível do sistema operacional hospedeiro, fator fundamental determinante de sua transportabilidade.

Como foi mencionado acima, utilizamos a estrutura conhecida como árvore B+ na implementação. As árvores B+ trabalham com unidades de acesso a memória secundária chamadas páginas. Uma página pode conter índices ou, quando localizada em uma folha, registros. A página corresponde a uma área de tamanho fixo, um segmento do arquivo.

Tendo em vista as características acima mencionadas do MAI, dividimo-lo em duas partes distintas, dois níveis de construção:

1. Método de Acesso Paginado, responsável pela manipulação de páginas pertencentes a um arquivo, em memória principal ou em disco.
2. Método de Acesso Indexado propriamete dito, utilizando-se das primitivas oferecidas pelo Método de Acesso Paginado e de árvores B+.

Graficamente, teremos vários ambientes, em sucessivas camadas ou níveis, dispostos da seguinte maneira:



De forma a atender os requisitos do MAI, o Método de Acesso Paginado deverá oferecer as seguintes primitivas:

a) Criar um arquivo paginado;
b) Abrir um arquivo paginado;
c) Fechar um arquivo paginado;
d) Alocar uma página;
e) Liberar uma página;
f) Ler uma página;
g) Gravar uma página;
h) Fixar uma página; e
i) Desfixar uma página.

O Método de Acesso Indexado deverá, por sua vez,

oferecer o conjunto de primitivas abaixo:

a) Criar um arquivo indexado;
b) Abrir um arquivo indexado;
c) Fechar um arquivo indexado;
d) Incluir um registro com uma chave única, inexistente;
e) Ler um registro dada uma chave já existente;
f) Alterar um registro;
g) Excluir um registro;
h) Posicionar o apontador de próximo registro em função de uma chave e de uma relação; e
i) Ler o próximo registro na ordem.

III) METODO DE ACESSO PAGINADO

O Método de Acesso Paginado tem como objetivo gerenciar a manutenção de uma quantidade de páginas de um arquivo em memória principal. Naturalmente, quanto maior o número de páginas em memória, maior a velocidade de acesso média alcancada pelo sistema. Visamos exatamente reduzir o tempo médio de acesso que depende diretamente do número de referências à memória secundária (um recurso caro).

Na criação ou abertura de um arquivo paginado se recebe um ponteiro, chamado identificador de arquivo paginado, que deverá ser usado para referenciar todo o acesso seguinte ao arquivo. Além disso, é alocada uma área de trabalho para conter as páginas que ficarão em memória e a tabela que controla estas páginas. Cada página da tabela tem associados a "hora" em que ocorreu o último acesso a ela, e o arquivo a que pertence. Além disso, como a página pode ser fixada na memória quando a aplicação assim o exigir, torna-se necessário também manter uma informação dizendo se a página e removível ou não.

A informação da "hora" do último acesso é usada na implementação do mecanismo de seleção para remoção da página menos recentemente utilizada (LRU em inglês). Isto permite que se libere a página que foi acessada pela última vez a mais tempo entre as removíveis. A liberação da página se dará quando for necessário espaço em memória para uma outra página lida ou criada (alocada). Existe um contador associado ao arquivo paginado que indica a "hora" atual do MAP e que é incrementado a cada acesso de leitura ou gravação.

As primitivas de fixação e desfixação de páginas na memória vêm atender, basicamente, as necessidades de manipular várias páginas na memória ao mesmo tempo. Observe que uma página só está garantidamente na memória desde após algum acesso a ela e até o próximo acesso a outra página qualquer, a menos que ela esteja explicitamente fixada.

Para contornar a incapacidade do CP/M de receber de volta as áreas em disco previamente alocadas porém não mais utilizadas o MAP deverá, também, gerenciar uma pilha com as páginas liberadas, de onde serão retiradas quando se tornar necessário, isto é, na alocação de páginas.

A utilização do Método de Acesso Paginado se dá

através de dez rotinas para o usuário e mais rotinas internas ao MAP. As primeiras nove rotinas correspondem exatamente às primitivas oferecidas e previstas no projeto do MAP, sendo que uma outra foi colocada posteriormente visando dar mais uma facilidade: ela retorna o número da página dado o seu endereço na memória.

IV) METODO DE ACESSO INDEXADO

O objetivo do Método de Acesso Indexado é fornecer um conjunto de rotinas ao sistema de aplicação para a manipulação de registros de tamanho variável a serem recuperados por uma chave única, também de tamanho variável.

Muitos algorítmos existem para a manipulação de árvores B (12). As árvores B+, porém, são mais complexas do ponto de implementação e muito cuidado deve ser tomado. Pouco pôde ser encontrado na bibliografia que auxiliasse na sua implementação. Descrevemos abaixo as características do nosso sistema.

O MAI é construído utilizando-se das primitivas do MAP. Sua única ligação com o meio físico é através do parâmetro que define o tamanho da página. Conseguiu-se, desta forma, também, uma independência neste nível em relação ao Sistema Operacional.

Basicamente, existem dois tipos de páginas no MAI: as páginas de índice e as páginas folha. As primeiras são responsaveis pelo armazenamento dos índices que apontam em última análise para as páginas do tipo folha. Um arquivo vazio não possui páginas índice. As páginas folha são as que contém os registros. Elas formam uma lista duplamente encadeada utilizada para a pesquisa sequencial.

Um registro de tamanho variável, como utilizado nesta implementação, tem no primeiro byte seu tamanho total. Imediatamente após este byte, segue-se o campo da chave de acesso, e, em seguida vêm os demais campos, se houver. Observe que esta escolha limita o tamanho total de registro a 256 bytes, o que é mais do que suficiente para as aplicações pretendidas. A alteração deste parâmetro, contudo é trivial.

A chave de acesso do registro, por sua vez, também é um campo de tamanho variável. Assim sendo, o seu primeiro byte é utilizado para armazenar o tamanho total do campo e é seguido da chave propriamente dita. Sempre que falamos da chave de acesso como parâmetro de entrada ou no interior de um registro, estamos nos referenciando ao campo de tamanho variável que possui a chave real e o tamanho do campo.

Na criação e abertura, a raiz é fixada na memória com o objetivo de reduzir o número de acessos a memória secundária visto que todo acesso direto e feito através dela. Assim, se a árvore tiver altura quatro, por exemplo, serão necessários no máximo três acessos no caso de nenhuma página estar presente na memória.

Sob o CP/M e com páginas de 2 registros físicos de 128 bytes, o número máximo de páginas e de (2**15 - 1) o que implica em um total de 8 Mbytes. A altura da árvore está limitada, neste caso portanto a 16.

O procedimento de exclusão garante a árvore

tornar-se no pior dos casos uma árvore binária balanceada. Se a árvore tiver altura H teremos no mínimo, portanto, 2 ** (H-1) registros podendo acessá-los em um máximo de (H-1) acessos. Na exclusão a concatenação só será efetuada quando a página estiver totalmente vazia. A subárvore que ficar pendente é reincluida em uma página "irmã" (de mesmo nível). Conseguimos, com isso diminuir a complexidade da operação de exclusão, que prevê a redistribuição de índices ou registros nas baixas ocupações das páginas. Naturalmente, tal decisão, implica em um comprometimento com a utilização de espaço em memória secundária, mas por outro lado diminui a frequência de alterações na árvore. No caso de nossas aplicações, onde a exclusão ocorrerá raramente não se perderá muito.

Num caso típico, conforme levantamentos os índices teriam por volta de 10 bytes o que nos permitiria ter 20 índices por página. Se considerarmos cada página com pouco mais da metade ocupada, digamos 12 índices, isto significa que cada página índice teria 13 filhos. Da mesma forma, se a árvore tiver uma altura H teremos 13 ** (H-1) folhas. Se cada folha contiver tipicamente 2 registros então poderemos acessar 2 * (13 ** (H-1)) registros em H-1 acessos. A tabela abaixo mostra alguns valores para cada altura H:

H	número médio de índices por página			número máximo de acessos
	1	9	19	
1	2	2	2	0
2	4	20	40	1
3	8	200	800	2
4	16	2000	16000	3
5	32	20000	320000	4

Na inclusão a quebra será efetuada quando a página não puder supertar a inclusão, pois a página é de tamanho fixo. O conteúdo da página será dividido com um nova página alocada e a inserção então realizada na página conveniente. A divisão será feita da seguinte forma:

- na quebra de folhas, a chave separadora do nível acima (índice) será o prefixo mínimo (13) entre a última chave da página anterior e a primeira chave da página seguinte. O último caracter da chave separadora será 0x00 se a última chave estiver contida na primeira, caso contrário será o sucessor do caracter pertencente à primeira chave que os difere;
- na quebra de índices um dos índices da página será escolhido para separar as duas páginas. Seu filho a direita será o primeiro filho à esquerda da nova página.

O emprego de técnicas de compactação para o armazenamento dos prefixos mínimos que formam os índices, poderia aumentar a densidade dos mesmos por página e, por conseguinte,

diminuir a altura da árvore de acesso.

Implementou-se apenas a primitiva que define uma pesquisa sequencial em ordem ascendente. Como, porém, as folhas estão duplamente encadeadas, também seria possível uma pesquisa sequencial em ordem contrária.

Uma página do tipo índice tem a seguinte disposição interna:

NOME DO CAMPO	TAMANHO EM BYTES	TIPO
1) CONTROLE	1	BINARIO
2) APONTADOR DE AREA UTIL (OFFSET)	2	NUMERICO BINARIO
3) PRIMEIRO FILHO	2	NUMERICO BINARIO
4) AREA DE TRABALHO	251	

Uma página do tipo folha, por sua vez, tem a seguinte disposição interna:

NOME DO CAMPO	TAMANHO EM BYTES	TIPO
1) CONTROLE	1	BINARIO
2) APONTADOR DE AREA UTIL (OFFSET)	2	NUMERICO BINARIO
3) FOLHA ESQUERDA	2	NUMERICO BINARIO
4) FOLHA DIREITA	2	NUMERICO BINARIO
5) AREA DE TRABALHO	249	

As páginas podem, ainda, ser raiz ou não. Usa-se os 2 bits menos significativos do byte de controle da página para identificá-las.

b0	INDICE	1
	FOLHA	0
b1	RAIZ	1
	NO	0

Assim temos 4 tipos de página, FOLHA (00), INDICE (01), UNICA (10), e RAIZ(11). A página única é raiz e folha.

V) CONCLUSOES

Atendendo aos objetivos, como se pode constatar, o trabalho possui as características essenciais de um projeto estruturado: independência entre os níveis, transportabilidade, modularidade, generalidade, confiabilidade, eficiência e simplicidade.

As rotinas são transportáveis, a menos da alteraçãao de algumas constantes, pelo fato de terem sido escritas na linguagem C, que vem se tornando mais comum a cada dia. Podem rodar prontamente em qualquer máquina que possua CP/M-

8080 ou CP/M-8086 (boa parte dos micros existentes) sem modificações. Como exemplo, salientamos que o sistema foi desenvolvido em vários microcomputadores (Polymax 201 e 301, Cobra 305, Sistema 700 da Prológica, e nos desenvolvidos no LESC).

O sistema é modular, com rotinas totalmente parametrizadas e independentes uma da outra. Essa modularidade permite:

- uma boa documentação;
- modificação de uma rotina sem ter que alterar as demais;
- o teste de cada rotina individualmente
- a possibilidade de estender o sistema para outras aplicações. (Por exemplo, representação de arquivos invertidos, fazendo-se com que cada folha armazene o conjunto de endereços de registros correspondentes a uma lista invertida).

Podemos definir um sistema de propósito geral (generalidade), como sendo um que pode ser utilizado para uma enorme variedade de aplicações. Nosso sistema, satisfaz esta característica também. Praticamente qualquer sistema de informação pode ser implementado utilizando as primitivas do MAI.

As rotinas foram escritas em C, o que nos garante uma boa confiabilidade contra falhas de software. Além disso, as rotinas foram testadas percorrendo todos os desvios. Embora o projetodo sistematenha sido "top-down", a implementação e os testes foram feitos "bottom-up".

Pretende-se que as rotinas sejam acessíveis por outras linguagens também, o que tornaria o MAI bem mais interessante comercialmente: BASIC e COBOL seriam os primeiros passos.

Uma das limitações do nosso sistema é a ausência de mecanismo contra falhas fisicas (p. ex., desligamento da energia). Poderíamos acrescentar técnicas de recuperação dinâmica de falhas. Porém, estas técnicas se mostram fracas diante das limitações do CP/M com respeito a integridade dos arquivos em tal situação.

Quanto a eficiência, sabemos que a linguagem C gera um bom código e o sistema de árvores B+ e sem dúvidas um dos mais eficientes métodos de acessos existentes. Porém, o nosso sistema pode se tornar mais eficiente com algumas medidas como:

- reprogramar em linguagem de montagem (assembly) o treche do código que gera uma quebra (e em consequência requer uma reorganização da árvore) (11).
- determinar automaticamente quantas páginas estarão na memória, de forma a maximizar a utilização do espaço disponível e, portanto, minimizar o número de acessos a memória secundária.

Finalmente, podemos dizer que o sistema é muito simples para o usuário ou o programador do nível de aplicação

utilizar. Basta possuir a biblioteca contendo as primitivas do MAI e conhecer suas definições (14). Como exemplo de utilização do MAI podemos citar um sistema de informação desenvolvido na PUC para controlar os estoques de circuitos integrados do LESC (14).

Concluindo, ratificamos que o objetivo do trabalho foi plenamente alcançado e esperamos que o sistema venha a ser de grande utilidade.

REFERENCIAS BIBLIOGRAFICAS

(1) - Rector R. e Alexy G., "The 8086 book", Osborne-McGraw-Hill, 1980.

(2) - "Osborne CP/M users guide", Osborne/McGraw-Hill, segunda edição, 1982.

(3) - Kernigham, B. W. e Ritchie, M., "The C programming language", Prentice Hall, 1978.

(4) - Date, C. J. "An Introduction to Data Base Systems". Addison-Wesley Publishing Co. Mass., 1977, Second Edition.

(5) - Furtado, A. L. e Santos, C. S. "Organização de Banco de Dados". Editora Campus Ltda. RJ, 1982.

(6) - Melo, Rubens N. "Estruturas de Armazenamento de dados". CCE, Depto. de Informática. PUC/RJ, 1981.

(7) - Wedekind, H. "On the selection of access paths in a data base system", Proc. IFIP Working Conference on Base Management, North Holland, 1974.

(8) - Comer, D., "The ubiquitous B-Tree", ACM Computing Surveys, 11(2), 6/79.

(9) - Bayer, R. e McCreight, C., "Organization and maintenance of large ordered indexes", Acta Informatica, 1, 3/72, pp 173-189.

(10) - Menascé, D. e Landes O., "On the design of a reliable storage component for distributed database management systems", Proc. Very Large Data Base Conference, Montreal, 1980.

(11) - Kruglinski, David "Data Base Management Systems". Osborne, McGraw-Hill, 1983.

(12) - Stonebraker, Michael. "Retrospection on a Data-Base System", ACM Transactions on Database Systems, 5(2), 6/80.

(13) - Bayer, R., Unterauer, K., "Prefix B-Trees" ACM Transactions on Data Base Systems, 2(1), 3/77,

(14) - Vieira, Ewerton A. P.,"Sistema de Controle de Estoque para Circuitos Integrados", Trabalho de Final de Curso em Engenharia Elétrica/Sistemas, PUC/RJ, 1983.

(15) - Menascé, D. e Landes, O., "Dynamic crash recovery of balanced trees", IEEE Computer Society Press, Symposium on reliability in distributed software, Pittsburgh. 7/81.