

Estruturas de Dados Específicas X Memória Transacional

Programação Concorrente e Paralela

Setembro de 2009



Estruturas de Dados para Paralelismo

- listas e filas (aula passada)
- tabelas de hash
 - paralelismo natural
 - locks com diferentes granularidades
 - listas livres de bloqueio
- skiplists



- motivação: *compositionality*
- como tornar atômicas operações que envolvem diferentes objetos?
 - necessidade de soluções *ad hoc* e de conhecimento prévio dos objetos manipulados
- dificuldade de esperar em mais que uma condição

Memória Transacional

- uso de construções similares às de transações de bancos de dados
- transações *linearizáveis*
- extensão de Java (descrita por Herlihy&Shavit):

```
public void enq (T x) {  
    atomic {  
        if (count == items.length)  
            retry;  
        items [tail] = x;  
        if (++tail == items.length)  
            tail = 0;  
        ++count;  
    }  
}
```



- composição de ações

```
atomic {  
    x = q0.deq();  
    q1.enq(x);  
}
```

- construção *orElse*: espera em alguma condição

```
atomic {  
    x = q0.deq();  
} orElse {  
    x = q1.deq();  
}
```



```

public boolean insert(int v)
    throws TMException {
    List newList = new List(v);
    TMOBJECT newNode = new TMOBJECT(newList);
    TMThread thread =
        (TMThread)Thread.currentThread();
    while (true) {
        thread.beginTransaction();
        boolean result = true;
        try {
            List prevList =
                (List)this.first.open(WRITE);
            List currList =
                (List)prevList.next.open(WRITE);
            while (currList.value < v) {
                prevList = currList;
                currList =
                    (List)currList.next.open(WRITE);
            }
            if (currList.value == v) {
                result = false;
            } else {
                result = true;
                newList.next = prevList.next;
                prevList.next = newNode;
            }
        } catch (Denied d){}
        if (thread.commitTransaction())
            return result;
    }
}

```



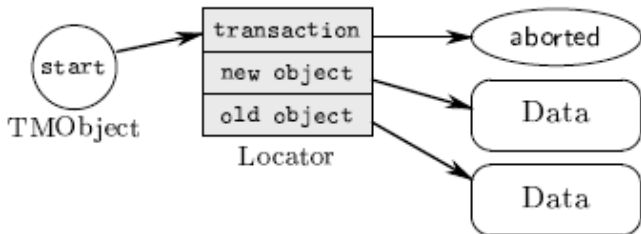
Memória Transacional – Implementação

- implementações por hardware: propostas baseadas em protocolos de coerência de cache
- implementações por software:
 - uso de descritores para objetos atômicos quando tentar de novo?
- na prática, construções como retry são bastante complicadas
 - problemas semelhantes aos dos monitores com testes implícitos: quando tentar de novo?

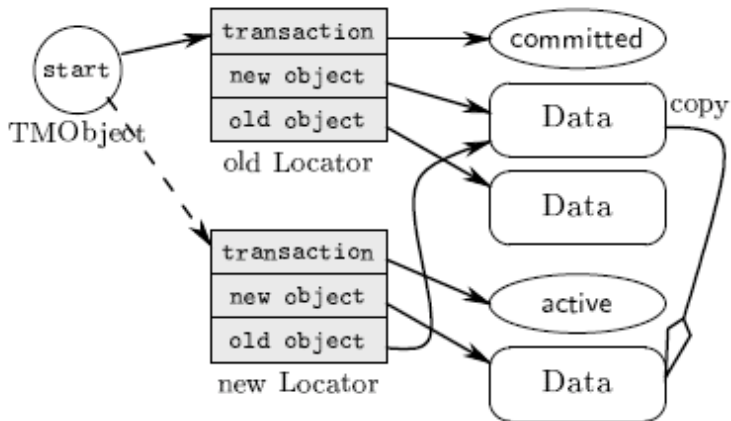


Objetos Atômicos – Indireção

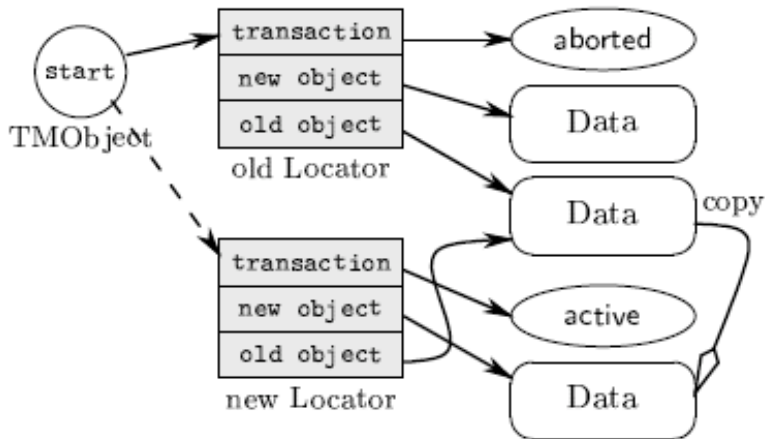
- descritor de objeto atômico:



Objetos Atômicos – transação comitted



Objetos Atômicos – transação abortada



Objetos Atômicos – transação ativa

- o que acontece se uma transação encontra um objeto com transação ativa?
- H&S propõem o uso de um gerente de transações com chamada *shouldAbort*
 - encapsulação de políticas como *backoff*, idade de transações, etc



- M. Herlihy, N. Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufmann, 2008.
- Maurice Herlihy, Victor Luchango, Mark Moir, and III William N. Scherer. Software transactional memory for dynamic-sized data structures. PODC 2003: 92-101, Jul 2003.
 - ler esse artigo e fazer um resumo para 1/10
 - 1 que problemas se tenta resolver com memória transacional?
 - 2 em que nível de programação se “enxerga” a MT (programador de aplicação, programador de bibliotecas, ...)?
 - 3 o que os autores propõem, em linhas gerais?
 - 4 o que se obteve com a proposta dos autores?
 - 5 quais são as maiores dificuldades de implementação?