

Uma Linguagem de Domínio Específico Baseada em Mônada com Suporte à Memória Transacional Distribuída em Java

Jerônimo da Cunha Ramos
Prof. Dr. André Rauber Du Bois (Orientador)
Prof. Dr. Maurício Lima Pilla (Co-orientador)

Mestrado em Computação
Programa de Pós-Graduação em Computação
Universidade Federal de Pelotas
`jdcramos@inf.ufpel.edu.br`

Abril de 2015



- 1 Memórias Transacionais**
- 2 Arquiteturas Distribuídas**
- 3 Trabalhos Relacionados**
- 4 Transações Distribuídas**
- 5 Transações Distribuídas**
- 6 Considerações Finais**



Memórias Transacionais

Introdução

- Alternativa à sincronização por locks;
- Similar às transações de bancos de dados (Atomicidade, Consistência, Isolamento);
- **Vantagens:**
 - Composabilidade, ausência de *deadlock*, facilidade de programação, maior exploração do paralelismo.
- **Desvantagens:**
 - Dificuldade em ter efeito colateral dentro de transações, *livelock* ou *starvation*.



Arquiteturas Distribuídas

Principais Diferenciais

- Cada nodo tem sua memória e espaço de endereçamento;
 - Impossibilita memória compartilhada;
 - Usa-se troca de mensagens;
- Problema do relógio global;
- Diferentes latências para acesso, devido à rede;
- Possíveis problemas ampliados: (exemplo: *deadlocks* distribuídos)
- Memória transacional também é um modelo promissor para este cenário.



Trabalhos Relacionados

CMTJava

- Estende Java para utilização do modelo transacional;
- Baseada na STM Haskell;
- Objetos Transacionais (implements TObject);
- Compilador cria métodos get e set especiais;
 - contêm o controle transacional;
 - retornam ações transacionais que só podem ser executadas dentro de blocos atômicos;
- Utiliza mônada de passagem de estados para passar o estado da transação entre as operações;



Trabalhos Relacionados

CMTJava

```
public STM<stm.Void> deposit (Double n) {
    return new STMDO{
        Double balance <- this.getBalance();
        this.setBalance(balance + n);
        Integer ops <- this.getOps();
        this.setOps(ops+1)
    };
}
```

Java

```
public STM<stm.Void> deposit (Double n) {
    return STMRTS.bind( this.getBalance(), { Double balance =>
        STMRTS.then( this.setBalance(balance + n),
        STMRTS.bind( this.getOps(), { Integer ops =>
            this.setOps(ops+1)
        }) ) }) ;
}
```

Trabalhos Relacionados

Focados em Arquiteturas Distribuídas

- Cluster-STM:
 - PGAS (*Partitioned Global Address Space*);
 - 4 combinações de estratégias implementadas.
- DiSTM:
 - Sem DSM;
 - 3 protocolos implementados.
- TFA:
 - Sem DSM;
 - Move os objetos;
 - Relógio Distribuído;



Transações Distribuídas

Transações Distribuídas

- A implementação atual da CMTJava contempla apenas transações locais;
- Pretende-se estender para transações distribuídas;
- Baseando-se no algoritmo TFA;
- Após a extensão, alguns objetos poderão ser movidos pela rede (implements RemoteTObject);
- É necessário algum tipo de diretório, para localização dos objetos



Transações Distribuídas

Exemplo de Código

```
//algum nodo registra o objeto em seu diretório
Directory.registry("account1", account);

//qualquer nodo abre o objeto e executa uma transação
account = (Account) Directory.openObject(address, "account1");
STMRTS.atomic(account.deposit(VALUE));
```



Algoritmos

OpenObject

```

Object openObject(String address, String id){
    if( éLocal(id) )
        return(lista.get(id).toObj);

    //envia requisição
    out.writeObject(new Request(LOOKUP, id, VersionClock.getReadStamp()));

    //recebe resposta
    resposta = (Request)in.readObject();
    if(resposta.getTipo() == OBJECT){
        //compara stamp local e remoto, conforme TFA
        if(VersionClock.getReadStamp() < resposta.getNodeStamp()){
            VersionClock.setStamp(resposta.getNodeStamp());
        }
        return (RemoteToObject)resposta.getMsg();
    }
}

pedido = entrada.readObject();
switch(pedido.getTipo()){
    case LOOKUP:
        //copara stamp local com o do remetente, conforme tfa
        if(VersionClock.getReadStamp() < pedido.getNodeStamp()){
            VersionClock.setStamp(pedido.getNodeStamp());
        }

        //envia objeto
        out.writeObject(new Request (OBJECT, lista.get(id); VersionClock.getReadStamp()));

        break;
}

```



Algoritmos

Commit

```
boolean commit(){  
    if(!adquire locks dos objetos locais e remotos no write set ){  
        releaseLocks();  
        return false;  
    }  
  
    Long writeVersion = VersionClock.getWriteStamp();  
  
    if (!valida conjunto de leitura){  
        releaseLocks();  
        return false;  
    }  
  
    Atualiza os valores no objeto local  
  
    releaseLocks();  
  
    for(String id : objetosRemotosNoWS){  
        Directory.changeOwner(id);  
    }  
  
    return true;  
}
```



Considerações Finais

Trabalhos Futuros

- Concluir implementação;
- Benchmarks;

Considerações Finais

- Memórias Transacionais tornam a programação paralela mais fácil;
- Este trabalho busca estender o conceito para os sistemas distribuídos;



Uma Linguagem de Domínio Específico Baseada em Mônada com Suporte à Memória Transacional Distribuída em Java

Jerônimo da Cunha Ramos
Prof. Dr. André Rauber Du Bois (Orientador)
Prof. Dr. Maurício Lima Pilla (Co-orientador)

Mestrado em Computação
Programa de Pós-Graduação em Computação
Universidade Federal de Pelotas
`jdcramos@inf.ufpel.edu.br`

Abril de 2015

