第33卷 第17期 计 算 机 工 程 2008年9月
Vol.33 No.17 Computer Engineering September 2007
·博士论文· 文章编号：1000—3428(2007)17—0010—02 文献标识码：A 中图分类号：TP311

分布式元组空间协同模型的设计与描述

黄永忠[1]，陈左宁[2]，周 蓓[1]，王 磊[1]
(1. 解放军信息工程大学信息工程学院，郑州 450002；2. 江南计算技术研究所，无锡 214083)

摘 要：元组空间是一种结构化的分布式共享存储编程方式，使用一个共享的元组空间进行生成式通信。该文针对集中式元组空间的性能瓶颈、单点失效问题及可扩展性差等不足，提出一种混合式的分布式元组空间的体系架构，给出基于π-演算的分布计算演算模型，并对模型的语法和操作语义进行讨论，通过钩互模拟给出了系统的性质。

关键词：分布式元组空间；π-演算；钩互模拟；位置

Design and Description of Distributed Tuplespace Coodinate Model

HUANG Yong-zhong[1], CHEN Zuo-ning[2], ZHOU Bei[1], WANG Lei[1]
(1. Information Engineering Institute, PLA Information Engineering University, Zhengzhou 450002;
2. Jiangnan Institute of Computing Technology, Wuxi 214083)

【Abstract】 Tuplespace is a kind of structural distributed shared memory programming paradigm. In this paradigm, a shared tuplespace is used to carry out generative communication. To improve the flaw of centralized tuplespace, such as performance bottle-neck, single failure, weak scalability, this paper puts forward a mixed distributed tuplespace architecture, presents the distributed computation model based on π-calculus, introduces the syntax and operational semantics of the model, shows some features of system through the barbed bisimulation.

【Key words】 distributed tuplespace; π-calculus; barbed bisimulation; location

1 概述

元组空间使用持久对象代替瞬时通信，其主要特点是生成式通信[1](generative communication)。进程之间的通信既不是通过消息传递，也不使用共享变量，而是使用一个共享的元组空间。进程可以向这个空间写入或者读出数据元组。数据元组是没有标记的，在决定一个读操作到底获得哪个元组时，使用一种基于元组内容的匹配机制。这样就使通信的各方互相匿名(anonymous)，而且基于元组空间的互进程通信是异步的，元组的生产者(producer)和消费者(consumer)不需要同步，从而实现时间去耦(time uncoupling)、目的去耦(destination uncoupling)和空间去耦(space uncoupling)。

和其他集中式系统一样，集中式元组空间最大的不足是性能瓶颈、单点失效问题及可扩展性差等问题，因此，开展了许多支持多个元组空间的研究[2]，将元组空间分布在网络环境中的计算结点使得通信双方可以在本地的元组空间直接进行，这样就可以解决通信瓶颈等问题，以适应基于模块化和层次式组织结构等构建较大规模软件系统。

本文给出了一种混合式的分布式元组空间的体系架构。在此基础上提出了基于π演算的分布计算演算模型，讨论了模型的语法和操作语义，并通过钩互模拟给出了系统的一些性质。

2 全分布与部分分布

根据元组空间的分布情况，全局共享元组空间可以是全分布或者部分分布。

全分布的实现在系统的每个结点都部署元组空间，每个元组根据某种分布策略分布在系统的某个结点上，这样可以保证系统有较高的容错性，但这需要有效的广播通信机制，因为需要查询元组空间的所有分区。

部分分布则选择若干结点作为单独的元组空间服务器，这一组元组空间服务器向外提供单一映像的元组空间访问服务，这种实现本质上只是提高原来的单一结点元组空间服务器的性能和可靠性，对于用户在使用上没有任何差别。

3 位置感知与位置透明

从位置透明性角度来看，分布式元组空间的设计时可以考虑位置感知(location awareness)和位置透明(location transparency)。

位置透明基于全局逻辑共享的思想，采用多个分布的元组空间来实现全局一致的透明元组空间。从应用的角度来看，用户感觉不到多个元组空间的存在，因此，系统提供位置透明性。

还有一种就是所谓带位置(located linda)的分布式元组空间，JavaSpaces, Tclaim 等都支持带位置的元组空间。这种模式可以使得通信本地化，每个结点都有一个自己的元组空间，其中系统中的进程和元组都标明位置。这种语义突出了多个不相交元组空间(disjoint tuple spaces)的概念，降低了位置透明性和空间耦合性。但如程序设计历史上关于 GOTO 语句的争论一样，某种程度上这种透明性的降低可以带来分布式程序设计灵活性的大大增强。

4 混合式元组空间

为了适应多种类型应用问题的求解，提供多范型支持，本文采用如图 1 所示的混合模型。

作者简介：黄永忠(1968－)，男，博士研究生，主研方向：分布与并行处理，软件工程；陈左宁，教授级高工、博士生导师、中国工程院院士；周 蓓、王 磊，讲师

收稿日期：2006-10-20 **E-mail：**yongzhong@263.net

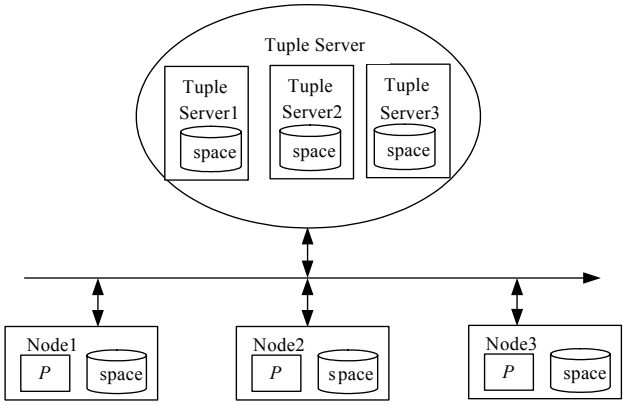

图 1　混合式分布式元组空间

其中，TupleServer 提供全局共享的元组空间，对于任务间无通信的问题，如简单任务包问题，可以通过 TupleServer 作为协同媒体实现协同。同时每个结点还有自己的元组空间，为结点间任务的直接通信提供支持。

系统存在 2 类元组空间：(1)全局共享的持久元组空间，为全局的任务分配和结果回收、持久对象保存等提供支持。(2)结点上的分布元组空间，为结点进程间的直接通信、协同提供支持。

5　分布式元组空间 π 演算

下面对于上节的混合模型用扩展的 π 演算模型[3-4]加以描述，其中最关键的扩展就是增加位置特性[5]。

5.1　基本语法

在本演算中，位置是表示分布性质的一种方式，其代表进程和资源存在的空间。每个计算结点都有一个位置标识，驻留在其上的进程和元组空间与该标识唯一绑定。唯一的例外是由一个多结点组成的全局（或称全域）元组空间服务器，该服务器对外提供单一映像，逻辑上相当于一个结点，其位置名为 Tupleserver。

Nets: N::=nil| l ::C | $\{l_1 \leftrightarrow l_2\}$| (vl)N | $N_1||N_2$
Componets: C::=<t >| P | $C_1|C_2$
Tuples: t::=u | t_1,t_2
Templates: T::=u | !x | T_1,T_2
Process: P ∷ = nil |<$S_1;S_2;\cdots;S_n$ > | a.P | P|Q | ！P
Actions: a::=in(T)@l | read(T)@l | out(x)@l
| move(Q, l) //表示代理(进程)Q 迁移到位置 l
| broadcast(N,t) //在网 N 上广播一条元组消息 t
| new(l) //创建一个新结点
| conn(u) | disc(u)

(1)网，用 *N,M,H,K*…等表示，由有限的结点集合及结点间的互连组成，每个结点有一个物理地址，为进程执行和数据驻留提供场所。*Xdπ*[GPS03]和 KLAIM 语言都提出了网的概念。一个结点是由位置名 *l*（结点的引用）及其位置上的组件 *C* 组成的一个对子。

(2)组件，用 *C,D*…等表示，可以是元组数据或进程，以及它们的组合。连接$\{l_1 \leftrightarrow l_2\}$表示 2 个位置之间是直接连接(或双向连接)。元组是一个顺序的名字序列。模板是用来匹配元组空间 TS(Tuple Space)的元组的模式。

(3)进程，用 *P,Q*…等表示，是模型中的计算单元，可以并发地在同一位置或不同位置上执行。可以是空进程（不活动进程）*nil*，和有动作前缀，也可以由多个进程复合而成。一个进程由多个顺序执行的策略<$S_1;S_2;\cdots;S_n$ >构成。

前缀操作允许进程读写元组数据，如 in(*T*)@*l*、read(*T*)@*l* 和 out(*x*) @*l*；可以移动到一个有名字的位置，move(*Q, l*)；在网络 *N* 上广播一条消息 *t*，broadcast(*N,t*)，广播是基于异步模式，即通过异步方式广播到网络 *N* 上的元组空间；new(*l*)创建一个新结点，conn(*u*) | disc(*u*)建立或断开一个连接。

注意，虽然 in(*T*)@*l*, read(*T*)@*l* 都是读取操作，但两者是有区别的。in(*T*)@*l* 操作查找位于 *l* 的 TS(Tuple Space)上匹配的元组<*t*>，如果与模板模式具有相同的域和域的数量，读取该数据，并将其从位于 *l* 的 TS 中移走；read(*T*)@*l* 类似于 in(*T*)@*l*，只是其仅读取位于 *l* 的 TS 上匹配的元组<*t*>，而不将其从元组空间移走。

5.2　规约语义

在(R-OUT)中，源结点和目的结点的连接关系存在是必需的，规则表明输出的执行将该操作的元组参数发送到目的结点的元组空间。(R-IN)和(R-READ)依赖于匹配函数 *match*(_;_)，该函数验证元组模板和相应的元组是否匹配，函数定义如下：

$$\frac{match(T_1;t_1)=\sigma_1 \quad match(T_2;t_2)=\sigma_2}{match(T_1,T_2;t_1,t_2)=\sigma_1\circ\sigma_2}$$

$match(l;l)=\epsilon$, $match(!x;l)=[l/x]$

其中，“ ϵ ”表示空替换；“ ◦ ”表示替换组合；“ $P\sigma$ ”表示将替换“ σ ”应用于进程 *P*。

(R-IN)动作 in(*T*)@*l'*可以消耗 *l'*中的与模式 *T* 匹配的数据<*t*>，读取之后<*t*>从 *l'*中的元组空间移走；(R-READ）中动作 read(*T*)@*l'*可以读取 *l'*中的与模式 *T* 匹配的数据<*t*>，读取之后<*t*>仍然在 *l'*中的元组空间。在（R-NEW）中，new(*l'*)的执行创建一个限定地址为 *l'* 的新结点，以及 *l* 和 *l'*之间的连接。(R-CREATE) 建立一个新进程 *Q*，建立之后 *Q* 与 *P* 并发执行。

定义 1 观察子或钩子：

$N \Downarrow< t >$若 $N \Rightarrow N'$ 并且 $N' \downarrow< t >$，这里 $\Rightarrow$ 是规约关系→的自反传递闭包。

定义 2 称网间的对称关系 $\mathfrak{R}$ 为弱钩模拟关系(weak barbed simulation)，记为 $N \approx \text{barb } M$，若对任意$(N, M)\in \mathfrak{R}$：

(1)只要 $N \Rightarrow N'$，必有 *M'*，使得 $M \Rightarrow M'$ 且 $(N',M')\in \mathfrak{R}$；

(2)对所有钩子< *t* >，若 $N \Downarrow< t >$，则有 $M \Downarrow< t >$。

定义说明，网间的对称关系 $\mathfrak{R}$ 为钩模拟关系(barbed simulation)，当 2 个网完成相同的归约动作，到达一致的可观察状态，访问相同的元组空间，则两者互模拟。

定义 3 称网间二元关系 $\mathfrak{R}$：

(1)钩保持(barb preserving)：若对任意$(N, M)\in \mathfrak{R}$，并且对所有钩子<*t*>，若 $N \downarrow< t >$ 则有 $M \downarrow< t >$；

(2)规约闭合(reduced closed)：若对任意$(N,M)\in \mathfrak{R}$，$N \rightarrow N'$，必有 *M'*，使得 $M \rightarrow M'$，且$(N',M')\in \mathfrak{R}$；

(3)上下文闭合(context closed)：若对任意上下文 *C*[.]，$(N, M)\in \mathfrak{R}$ 隐含$(C[N], C[M])\in \mathfrak{R}$。

定义 4 称网间满足钩保持、规约闭合、上下文闭合的最大二元关系 $\mathfrak{R}$ 是强钩互模拟关系，记为 $N \sim \text{barb } M$。

定律 1 $\equiv \subset$ barb。

证明：$\subseteq$ 显然。证明 $\equiv$ 是 barb 的真子集，可给出一个例子，$a.nil \mid a.nil$ barb $a.a.nil$，但 $a.nil \mid a.nil \not\equiv a.a.nil$，$\not\equiv$ 表示不等价。

引理 1 令 P^i 及 Q^i ($i \in J$)分别为一组进程。如果对于所

（下转第 14 页）