Математические вопросы кибернетики и вычислительной техники 20, 1998, 39–44.

Построение полной системы примеров для распределенных программ в АКМ-сети

А. Ю. Шукурян

Институт проблем информатики и автоматизации НАН РА и ЕрГУ

Резюме

В настоящей работе предлагается вычислительная модель сети из автоматических кассирных машин (АКМ-сети) и вводится понятие распределенной программы над специальной системой команд, включающей обработки счетчиков и очереди. Программа состоит из множества клиентских программ, взаимодействующих с одной серверной программой. Исследуется проблема отладки для обеспечения корректного функционирования распределенной программы. Показано, что для достаточно широкого класса распределенных программ при ограничениях на глубину циклов в программе, проблема построения полной системы примеров (ПСП) алгоритмически разрешима.

1. АКМ-сеть

Основными компонентами которой являются АКМ, расположенные удаленно от основного компьютера и инициируемые клиентами. Каждый из клиентов имеет собственный счет, пароль, зафиксированные в основном компьютере. При инициации АКМ клиент по запросу от кассирной машины вводит последовательно пароль, счет и тип услуги, которую он хочет получить. После ввода услуги АКМ запускает (стартует) выполнение сделки в основном (серверном) компьютере и ожидает возврата с результатом качества выполненной сделки. Обращение к серверному компьютеру для выполнения сделки происходит многократно от различных АКМ. Соответственно в общей памяти серверного компьютера возникает очередь выполняемых сделок, которая обрабатывается последовательно. Обработка завершается, если очередь пуста. Возникновение очереди, в частности, может быть вызвано выполнением сервером других работ, отличных от выполнения сделок. Будем считать для упрощения, что до завершения выполнения текущей сделки новая сделка не выполняется.

С применением простейших программных конструкций управление АКМ может быть представлено с помощью следующего псевдокода (рис.1).

```
do forever
        высветить основной экран
        читать карту
        repeat
                запросить пароль
                читать пароль
                проверить пароль
        until  счет правильный
```

```
repeat
        repeat
                запросить тип сделки
                читать тип
                запросить сумму
                читать сумму
                стартовать сделку
                ждать выполнения сделки
        until  сделка успешна
        распределить наличные
        ждать взятия клиентом
        опросить продолжение
until  клиент просит остановить
выдать карту
ждать пока клиент заберет карту
```

Рис.1. Управление АКМ.

В приведенном описании подчеркнуты обращения к основному компьютеру, работа которого может быть описана следующим образом

```
do
        repeat
                проверить счет из очереди
        until   результат проверки счета возвращен
        repeat
                repeat
                        выбрать очередную сделку
                        выполнить сделку
                until результат выполнения сделки возвращен
        until  очередь сделок пуста
        exit
```

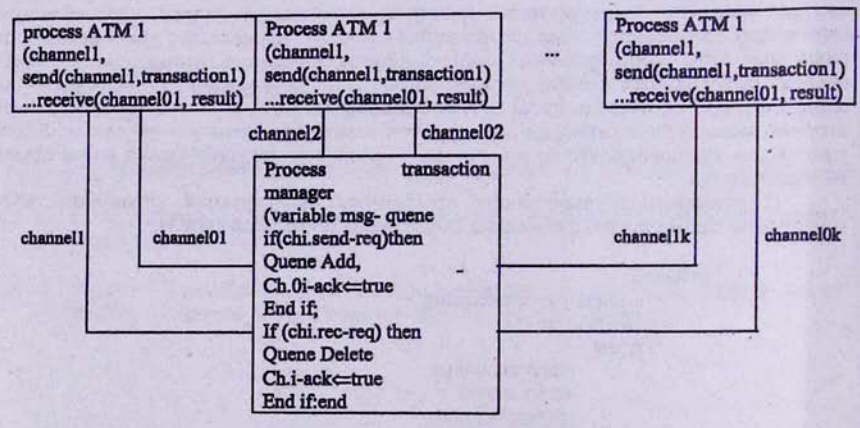

Рис.2. Схема взаимодействия АКМ и основного сервера.

Распределенные программы АКМ-сети постоянно расширяются за счет ключения новых клиентских возможностей, что приводит к многочисленным ррективкам и необходимости разработки формализованных методов отладки. вестен подход к решению этой задачи путем построения полной системы имеров. Ниже предлагается вычислительная алгоритмическая модель для сети ХМ. В идейном плане модель является специальным расширением модели машины рздиня [1], а предпринятое исследование в определенном смысле является одолжением работы [2].

Абстрактная машина имеет конечное число **входных лент** $X = \{X_1, X_2, ..., \}$, **очередь** Q (первый вошедший- первый выходящий), конечное число **счетчиков** Z $\{Z_1, Z_2, ..., Z_n\}$. Очередь и счетчики являются подмножествами **совместно пользуемой памяти (СИП)**. В СИП включается также множество совместно пользуемых внутренних ячеек $V=\{v_1, v_2, ..., v_i\}$. Помимо СИП в абстрактной ашине имеется k **локальных памятей** M_j, состоящих из внутренних ячеек $V_j = \{v_{1j}, , ...,v_{aj}\}$, j = 1,2,..., k. Таким образом абстрактная машина является ультипроцессором с центральным серверным компьютером.

Входные ленты разделены на ячейки, в каждой из которых может быть писано произвольное неотрицательное целое число. Каждая входная лента набжена одной считывающей головкой. Очередь разделена на ячейки и снабжена дной записывающей и одной считывающей головками.

Машина имеет следующие **команды:**

$v_{ij} \leftarrow C$ ($v_i \leftarrow C$)

о внутреннюю ячейку v_{ij} локальной памяти M_j (во внутреннюю ячейку v_i СИП) аписывается константа C .

$v_{ij} \leftarrow v_{ik}$ ($v_i \leftarrow v_j$) ($v_{ij} \leftarrow v_k$)

о внутреннюю ячейку v_{ij} локальной памяти M_j (во внутреннюю ячейку v_i СИП) аписывается содержимое внутренней ячейки v_{ik} локальной памяти (внутренней ячейки v_j СИП).

$Q \leftarrow v_{ij}$

оловка очереди Q сдвигается на одну ячейку вправо и в обозреваемую ячейку аписывается содержимое внутренней локальной памяти v_{ij}.

рименительно к случаю АКМ-сети интерпретация следующая: в очередь ересылается тройка (i, y, l), где I- номер АКМ, y- код сделки , l- номер чета(счетчика). Код сделки имеет следующую интерпретацию:

сли $y = Z_l$- 1, то содержимое Z_l уменьшается на единицу;

сли $y = (Z_l - 1 ; Z_l + 1)$, то содержимое Z_l уменьшается на единицу, а содержимое увеличивается на единицу.

$v_i \leftarrow Q$

Содержимое обозреваемой считывающей головкой ячейки очереди Q записывается о внутреннюю ячейку v_i. После чтения считывающая головка сдвигается вправо.

if empty(Q) then goto L_1 else goto L_2.

Проверка очереди Q на пустоту. Если очередь пуста, то выполняется переход на оманду с меткой L1; если очередь не пуста, переход осуществляется на меткуL2.

do forever... end

существляется циклическое выполнение группы команд, расположенных между do orever... end.

goto L.

Осуществляется переход на команду с меткой L.

if $v_{ij} \leftarrow X_j$ then goto L1 else goto L2

Содержимое обозреваемой ячейки входной ленты X_j записывается во внутреннюю ячейку v_i локальной памяти M_j. Если в обозреваемой ячейке ленты записано некоторое целое число, то после передачи числа во внутреннюю ячейку v_i головка

ленты сдвигается на одну ячейку вправо и осуществляется переход на команду с меткой L1; если обозреваемая ячейка ленты пуста, то сдвига головки не происходит, содержимое внутренней ячейки не меняется, переход осуществляется на команду с меткой L2.

9. **if** v_{ij} σ v_{ik} (v_i σ v_j) **then goto L1 else goto L2**, σ∈{>,<}.
Содержимое внутренней ячейки v_{ij} локальной памяти M_j (внутренней ячейки v_i СИП) сравнивается с содержимым внутренней ячейки v_{ik} локальной памяти (внутренней ячейки v_j СИП). При выполнении условия σ переход осуществляется на команду с меткой L1, иначе на команду с меткой L2.

10. **if** v_{ij} σ C (v_i σ C) **then goto L1 else goto L2**, σ∈{>,<}.
Содержимое внутренней ячейки v_{ij} локальной памяти M_j (внутренней ячейки v_i СИП) сравнивается с константой С При выполнении условия σ переход осуществляется на команду с меткой L1, иначе на команду с меткой L2.

11. **if** Z_i σ C **then goto L1 else goto L2**, σ∈{>,<}.
Содержимое счетчика Z_i СИП сравнивается с константой С При выполнении условия σ переход осуществляется на команду с меткой L1, иначе на команду с меткой L2.

12. $Z_i \leftarrow Z_i$ +1.
Прибавление 1 к счетчику.

13. $Z_i \leftarrow Z_i$-1.
Вычитание 1 из счетчика.

14. ($Z_i \leftarrow Z_i$ -1; $Z_i \leftarrow Z_i$ +1).
Групповая команда , имитирующая "перевод" единицы с одного "счета" на другой.

15. **exit.**
Команда завершения.

Клиентской программой назовем непустое множество команд (1, 2, 6, 7, 8, 9, 10), относящихся к единственной локальной памяти и не относящихся к СИП, счетчикам и очереди, включающее лишь команду (3) записи в очередь.

Серверной программой назовем непустое множество команд (1, 2, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15), не относящихся к ячейкам локальных памятей.

Стандартным образом каждая из вышеприведенных программ может быть представлена в виде ориентированного графа с вершинами- командами, отмеченными метками и дугами-переходами.

Распределенной программой назовем совокупность $P = (P_i, P_2, ..., P_k, P_0)$ k клиентских программ P_i (i =1, 2,..., k) (по числу входных лент машины) и одной серверной программы P_0. Считается, что локальные памяти клиентских программ не пересекаются.

Две клиентские программы будем считать **одинаковыми**, если их графы изоморфны и команды соответствующих вершин совпадают с точностью до переименования индексов переменных локальных памятей. Распределенная программа называется **однородной**, если все клиентские программы, входящие в нее, одинаковы.

2. Проблема построения полной системы примеров для однородных распределенных программ.

Примером для распределенной программы являются массивы целых чисел на входных лентах, в очереди, счетчиках и значения внутренних ячеек. Путем в распределенной программе называется вектор $\alpha = (\alpha_1, \alpha_2, ..., \alpha_k, \alpha_0)$, для которого α_i - последовательность команд, следующих друг за другом в соответствии с метками (или последовательность команд , следующих друг за другом в графе программы) в

клиентской программе P_1. Путь α_0 соответствует серверной программе P_2. Путь α называется **реализуемым**, если соответствующая программа проходит по соответствующей пути- компоненте вектора α. Вектор, составленный из подпоследовательностей (ветвей), принадлежащих компонентам реализуемого пути, называется реализуемым. Пример называется **допустимым** для распределенной программы, если программа, запущенная на этом примере, приводит к остановке серверной программы. конечное множество примеров Ω называется **полной системой примеров (ПСП)** для программы абстрактной машины, если каждый реализуемый вектор программы принадлежит пути, активизируемому некоторым примером из Ω. Говорят, что проблема ПСП для программы из некоторого класса программ Ψ разрешима, если существует алгоритм, который по любой программе P из Ψ строит ПСП для P.

Из [3] следует, что для подкласса LOOP(i) программ (распределенные программы не рассматривались) в системе команд (1-15) (i-количество допустимых уровней вложения команд типа do... end) при i = 2 проблема построения полной системы примеров алгоритмически неразрешима. Однако для LOOP(1) проблема ПСП разрешима. Основываясь на этом, в [4] был введен класс AL_Z программ в системе команд (1-15).

AL_Z - программой называется одноцикловая программа, составленная из любого непустого множества команд (1-15) со следующими ограничениями:

а/ вложенные циклы do forever ... end запрещены;

б/ команды с метками L1 и L2 расположены после команды if...then goto...else goto... и хотя бы одна из них не принадлежит циклу do forever...end.

В [4] доказана следующая теорема.

Теорема 1. [4]. Проблема построения ПСП разрешима в классе AL_Z программ.

Через DAL_Z обозначим класс распределенных однородных программ, для которых серверная программа является AL_Z -программой.

Основной результат формулируется следующим образом.

Теорема 2. Проблема построения ПСП разрешима в классе распределенных программ DAL_Z.

Заметим, что с учетом теорем 1 и 2 имеет место

Следствие. Проблема построения ПСП разрешима для подкласса программ из DAL_Z, состоящих из одной клиентской программы.

Доказательство теоремы 2 проведем индукцией по числу k клиентских программ. Итак, утверждение теоремы справедливо для k=1. Предположим, что оно справедливо для подкласса $DAL_Z(k-1)$ с k-1 клиентской программами.

Рассмотрим распределенную однородную программу $P_{k-1} = (P, P, ..., P(k-1 раз), P_0)$ из $DAL_Z(k-1)$. Пусть $\Sigma(P_{k-1})$- полная система примеров для программы P_{k-1} и $\sigma = (\sigma_1, \sigma_2, ..., \sigma_i, ..., \sigma_{k-1}, \sigma_0)$ принадлежит $\Sigma(P_{k-1})$. Рассмотрим вектор sub $(\sigma, \sigma_i) = (\sigma_1, \sigma_2, ..., \sigma_i, ..., \sigma_{k-1}, \underline{\sigma_i}\ \underline{\sigma_0})$, где $\underline{\sigma_i}$ определено на множестве внутренних ячеек v_{kj} ($v_{kj} \in M_k$), входной ленте X_k и $\underline{\sigma_i}(v_{kj}) = \sigma_i(v_{ij})$, $\underline{\sigma_i}(X_k) = \sigma_i(X_i)$. Компонента $\underline{\sigma_0}$ определена на множестве внутренних ячеек v_i ($v_i \in M$)из СИП, счетчиках Z: $\underline{\sigma_0}(v_i) = \sigma_i(v_i)$, $\underline{\sigma_0}(Z_s) = \sigma_0(Z_s)$. Пусть значение очереди Q на компоненте σ_0 примера σ будет $\sigma_0(Q) = d_1 d_2 ... d_s d_p$. . Выделим такое значение d_s, для которого $d_s(M) = v_{ij}$. Тогда $\underline{\sigma_0} = d_1 d_2 ... d_s ... d_p d_s$. Рассмотрев соответствующий вектор пути, легко показать, что вектор $\sigma = (\sigma_1, \sigma_2, ..., \sigma_i, ..., \sigma_{k-1}, \sigma_0)$ является примером для P_{k-1}.

Обозначим $\underline{\Sigma(P_k)} = \cup \cup \text{sub}(\sigma, \sigma_i)$.

Пользуясь доказательством от противного, легко убедиться, что $\underline{\Sigma(P_k)}$ является полной системой примеров $\Sigma(P_k)$ для программы P_k.

Литература

[1]. Я. М. Барздинь и др., Построение полной системы примеров для проверки программ. В сб. "Ученые записки Латвийского ГУ", 210, Рига,1974 г.

[2] А. Ю. Шукурян, О полной системе примерив для обнаружения конфликтов взаимодействующих программ. Математичеслие вопросы кибернетики и вычислительной техники,17, 1997,46-51.

[3] E. M. Gurary, Decidable problems for powerful programs. Journal of ACM.1985, 32, N 2, pp. 466-483.

[4] А. А. Саркисян, О проблеме построения полной системы примеров для одного класса простых программ. Докл. АН Арм. ССР. 1989, 88, N 3, стр. 108-111.