Распределённые алгоритмы

mk.cs.msu.ru → Лекционные курсы → Распределённые алгоритмы

Семинар 7

Алгоритмы маршрутизации

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, весенний семестр

Задача PIF (propagation of information with feedback) широковещательного распространения информации по сети с подтверждением завершения связи устроена так

Задан узел p, обладающий данными d

Требуется гарантированно (в т.ч. без бесконечных вычислений)

(PI) доставить данные d во все узлы сети и

(F) хронологически после этого уведомить узел p о том, что данные доставлены

- Уверенность узла p в том, что данные доставлены, обозначим командой done
- Требуемое уведомление узла p — это выполнение done после приёма всеми узлами сообщений с d

PIF-алгоритм — это распределённый алгоритм решения задачи PIF

Утверждение (Задача 1.1). Если $\mathrm{done} = \mathrm{decide}$, то PIF-алгоритм является волновым алгоритмом

Доказательство.

Свойства завершаемости и принятия решения PIF-алгоритма $\mathcal{A}$ выполняются по определению PIF-алгоритма

Осталось убедиться в полноте покрытия

Предположим от противного, что это свойство не выполняется

Это означает, что существуют вычисление E, действие принятия решения $\alpha \in \mathfrak{Act}(E, \mathcal{A}) = (\alpha_1, \alpha_2, \dots)$ и узел q, такие что в ни для одного действия β узла q не верно $\beta \preceq \alpha$

Тогда существует перестановка $\mathfrak{B}$ действий из $\mathfrak{Act}(E, \mathcal{A})$, сохраняющая причинно-следственный порядок и такая что все действия узла q располагаются хронологически позже действия α

По теореме о перестановке действий, существует и вычисление алгоритма $\mathcal{A}$, отвечающее действиям $\mathfrak{B}$, и в нём α (done) выполняется до первого действия узла q — то есть алгоритм $\mathcal{A}$ не является PIF-алгоритмом (*противоречие*) ▼

Утверждение (Задача 1.2). Любой централизованный волновой алгоритм, в котором решение принимает инициатор, может быть преобразован в PIF-алгоритм с теми же коммуникационной сложностью и сложностью по времени

Доказательство.

Это преобразование можно устроить такЖ

- Инициатору к каждому отправляемому сообщению добавить рассылаемые данные d
- Последователю
  - При приёме первого сообщения сохранить полученные данные d
  - Добавить к каждому отправляемому сообщению данные d
- Заменить decide на done

Тогда данные d будут доставлены в каждый узел-последователь при выполнении первого действия этого узла

Принятие решения волновым алгоритмом означает, что в каждом узле-последователе выполнено хотя бы одно действие

Следовательно, ко времени принятия решения данные d доставлены во все узлы сети ▼

Задача 1.3: преобразовать алгоритм эха в PIF-алгоритм с той же коммуникационной сложностью

Код инициатора p:

1. Для всех $q \in Neigh_p$: $\mathrm{send}_q(\mathbf{tok})$
2. $N_p := 0$;
3. Пока $N_p < |Neigh_p|$:
   3.1 $\mathrm{receive}_q(\mathbf{tok})$ для любого $q \in Neigh_p$
   3.2 $N_p := N_p + 1$;
4. decide

Код последователя p:

1. $\mathrm{receive}_q(\mathbf{tok})$ для любого $q \in Neigh_p$
2. Для всех $r \in Neigh_p \setminus \{q\}$: $\mathrm{send}_r(\mathbf{tok})$
3. $N_p := 0$;
4. Пока $N_p < |Neigh_p| - 1$:
   4.1 $\mathrm{receive}_r(\mathbf{tok})$ для любого $r \in Neigh_p$
   4.2 $N_p := N_p + 1$;
5. $\mathrm{send}_q(\mathbf{tok})$

Задача 1.3: преобразовать алгоритм эха в PIF-алгоритм с той же коммуникационной сложностью

Код инициатора p:

1. Для всех $q \in Neigh_p$: $\mathrm{send}_q(\textbf{pack}, d)$
2. $N_p := 0$;
3. Пока $N_p < |Neigh_p|$:
   3.1 $\mathrm{receive}_q(\textbf{pack}, d)$ для любого $q \in Neigh_p$
   3.2 $N_p := N_p + 1$;
4. done

Код последователя p:

1. $\mathrm{receive}_q(\textbf{pack}, d)$ для любого $q \in Neigh_p$
2. Для всех $r \in Neigh_p \setminus \{q\}$: $\mathrm{send}_r(\textbf{pack}, d)$
3. $N_p := 0$;
4. Пока $N_p < |Neigh_p| - 1$:
   4.1 $\mathrm{receive}_r(\textbf{pack}, d)$ для любого $r \in Neigh_p$
   4.2 $N_p := N_p + 1$;
5. $\mathrm{send}_q(\textbf{pack}, d)$

Задача SYN (SYNchronization) синхронизации узлов сети, устроена так

Для каждого узла q задано всегда допустимое внутреннее действие α_q

Задано непустое подмножество узлов P, и для каждого узла $p \in P$ — всегда допустимое внутреннее действие β_p

Требуется гарантированно (в т.ч. без бесконечных вычислений) выполнить все действия α_q и β_p, причём так, чтобы каждое действие α_q было выполнено хронологически раньше каждого действия β_p

SYN-алгоритм — это распределённый алгоритм решения задачи SYN

Утверждение (Задача 2.1). Каждый SYN-алгоритм можно перестроить в волновой алгоритм с теми же коммуникационной сложностью и сложностью по времени

Утверждение (Задача 2.2). Каждый волновой алгоритм можно перестроить в SYN-алгоритм (для подходящего множества P) с теми же коммуникационной сложностью и сложностью по времени

Точной нижней гранью элементов a и b ЧУМ $(X, \leq)$ называется элемент c этого множества, для которого верно следующее:

- $c \leq a$ и $c \leq b$ (*то есть это нижняя грань*)
- Для любого элемента d, такого что $d \leq a$ и $d \leq b$, верно $d \leq c$ (*то есть это нижняя грань, наиболее близкая к исходным элементам*)

Нижняя полурешётка — это ЧУМ, для любых двух элементов которого существует точная нижняя грань

Точную нижнюю грань элементов a, b нижней полурешётки обозначим записью $a \downarrow b$

Примеры точных нижних граней: min для множеств чисел; & для булевых значений; $\cap$ для множеств, если $\leq$ — это $\subseteq$

Известно, что точная нижняя грань в нижней полурешётке единственна, и операция её вычисления коммутативна $(a \downarrow b = b \downarrow a)$ и ассоциативна $(a \downarrow (b \downarrow c) = (a \downarrow b) \downarrow c = a \downarrow b \downarrow c)$

$\downarrow X$ — так для конечного непустого множества $X = \{a_1, \ldots, a_n\}$ обозначим точную нижнюю грань его элементов: $\downarrow X = a_1 \downarrow a_2 \downarrow \ldots \downarrow a_n$

Задача INF распределённого вычисления точной нижней грани значений нижней полурешётки $(X, \leq)$, располагающихся во всех узлах сети, устроена так

Для каждого узла p задан элемент a_p полурешётки

Задан узел $\mathfrak{p}$, и в нём содержится переменная *out* типа X

Требуется

- получить в *out* точную нижнюю грань всех элементов a_p сети и
- после этого выполнить в $\mathfrak{p}$ команду done завершения вычисления

INF-алгоритм — это распределённый алгоритм решения задачи INF

Утверждение (Задача 3.1). Каждый INF-алгоритм можно перестроить в волновой алгоритм с теми же коммуникационной сложностью и сложностью по времени

Утверждение (Задача 3.2). Каждый волновой алгоритм можно перестроить в INF-алгоритм с теми же коммуникационной сложностью и сложностью по времени и с битовой сложностью, превосходящей исходную битовую сложность не более чем в $O(w)$ раз, где w — количество битов для хранения одного элемента из X