УДК 004.9

В.В. Литвин, А.С. Мельник
Національний університет “Львівська політехніка”,
кафедра інформаційних систем та мереж

АНАЛІЗ МЕТОДІВ РОЗВ’ЯЗУВАННЯ ЗАДАЧІ ПЛАНУВАННЯ В ОБЧИСЛЮВАЛЬНИХ ГРІД-СИСТЕМАХ

© Литвин В.В., Мельник А.С., 2010

Планування виконання робіт є однією з найважливіших та найскладніших задач у грід-системах, оскільки ця задача є NP-повною. У статті визначено особливості організації планування у грід-системах; розглянуто основні алгоритми, розроблені для планування у грід.

Ключові слова: грід-системи, планування робіт, планування на основі best-effort підходу, планування на основі QoS підходу, гетерогенні системи, розподілені обчислення.

Workflow scheduling is one of the most important and most difficult issues in grid systems. Finding a best solution for a workflow scheduling problem is NP-complete. This paper considers specifics of workflow scheduling in grid systems and presents existing scheduling algorithms developed for grids.

Keywords: grid systems, workflow scheduling, best-effort based scheduling, QoS constraint based scheduling, heterogeneous systems, distributed computing.

Постановка проблеми та формування цілі дослідження

Грід-системи бувають різних розмірів і типів. Найпростіший грід складається з кількох машин однакового типу, з’єднаних локальною мережею. Цей тип гомогенної системи як грід розглядають рідко [1]: зазвичай її називають кластером і використовують переважно для випробування програмних засобів гріда [2]. Всі машини такої грід-системи знаходяться в межах одного відділу, тому її використання не вимагає наявності спеціальних політик і систем безпеки.

Складнішим грід є гетерогенна система, яка складається з машин кількох різних відділень однієї організації. В такому інтрагріді можуть використовувати певні політики (наприклад, пріоритетності відділень) та системи безпеки. Важливість системи безпеки зростає, якщо відділення організації географічно розкидані, знаходяться в різних містах і, наприклад, з’єднані у VPN через Інтернет.

Грід, який об’єднує гріди кількох різних організацій (інтергрід), вимагає наявності високого рівня систем безпеки для захисту від можливих атак та шпигунських програм [2]. Інтергрід можна використовувати як для спільних проектів (партнерські системи), так і в комерційних цілях (торгівля ресурсами).

У комерційному гріді: 1) метою провайдерів ресурсів є отримання максимальної віддачі від своїх інвестицій; 2) користувачі ресурсів намагаються виконати свої роботи в межах певного бюджету і часового обмеження. Ціна ресурсів може змінюватися з часом (згідно з законом попиту і пропозиції), дозволяючи найважливішим роботам виконуватися в час “пік”, а менш важливим – після спаду навантаження. В партнерських грід провайдери надають ресурси безкоштовно.

Більшість систем управління ресурсами грід-систем реалізовано відповідно до монолітної архітектури і важко переносяться на інші системи [3]. Будова та принцип роботи системи управління ресурсами залежать від особливостей грід-системи: менеджер ресурсів інтергріда може істотно відрізнятися від менеджера ресурсів інтрагріда, оскільки повинен враховувати політику кожного адміністративного домену; планування в партнерських грід та в комерційних грід будуть відрізнятися через різну цінову політику – у комерційних грід повинна бути встановлена економічно керована система управління ресурсами; розмір грід-системи може ставити додаткову вимогу щодо продуктивності алгоритмів планування тощо.

Метою дослідження є математична постановка задачі планування для гетерогенних грід-систем; визначення особливостей організації планування у грід-системах різного типу; огляд існуючих алгоритмів планування для визначення їх застосовності та ступеня задоволення вимог щодо планування у грід-системах різного типу.

Визначення основних понять

Нижче подамо визначення основних понять в контексті цієї роботи.

Грід (грід-система) – розподілена обчислювальна система, яка забезпечує гнучкий, безпечний, скоординований розподіл ресурсів. Грід може об'єднувати обчислювальні машини, розподілені як адміністративно, так і географічно.

Робота (задача, алгоритм) – обчислювальна одиниця, яку потрібно виконати у грід-системі. Загальна робота складається із взаємозв'язаних неподільних робіт меншого розміру. Неподільна робота повинна виконуватися на одній обчислювальній машині. Кожна робота має свої власні вимоги до ресурсів.

Ресурс (обчислювальна машина) – обчислювальна одиниця, надана провайдером ресурсів для використання у грід-системі для виконання робіт. Кожен ресурс має свої власні властивості, наприклад, потужність процесора, об'єм пам'яті, тип програмного забезпечення тощо.

Планування (розподіл робіт в мережі; диспетчеризація робіт) – визначення порядку виконання робіт та вибір ресурсів, на яких ці роботи будуть виконуватися.

Планувальник (диспетчер робіт) – складова грід-системи, яка виконує планування. У гріді можуть бути декілька планувальників, зокрема різного ієрархічного рівня.

Система управління ресурсами (система планування, менеджер ресурсів) – система, яка виконує планування та забезпечує виконання робіт у гріді. До цієї системи входять всі планувальники грід-системи.

Політика – констатація чітко визначених вимог, умов та пріоритетів, виставлених провайдерами ресурсів грід-системи.

Домен – множина ресурсів грід-системи, які є суб'єктом однієї або декількох політик.

Вимоги якості обслуговування (QoS-обмеження) – вимоги, пред'явлені користувачами грід-системи щодо виконання їхніх робіт.

Гомогенна система (однорідна система) – система, яка складається з обчислювальних машин з однаковими властивостями.

Гетерогенна система (різнорідна система) – система, яка складається з обчислювальних машин з різними властивостями.

Інтрагрід – грід, який складається з машин одного або декількох відділень однієї організації, зокрема географічно розподілених.

Інтергрід (глобальний грід) – грід, який об'єднує гріди декількох різних організацій.

Основний матеріал

Способи організації планування в грід-системах

Згідно з [3, 4] системи планування в грід можна класифікувати на централізовані, ієрархічні та децентралізовані.

У **централізованій системі** планування виконує єдиний диспетчер, який володіє повною інформацією про всі ресурси системи. Централізована система планування погано масштабується і може стати вузьким місцем великої грід-системи: задача планування є NP-повною і для знаходження оптимального розв'язку потрібні значні обчислювальні ресурси; а при збої планувальника роботу системи буде заблоковано. З іншого боку, володіючи повною інформацією, централізована система може виконувати планування дуже ефективно.

В інтергрідах машини розміщені в різних адміністративних доменах. Кожен адміністративний домен має свою власну політику і деякі домени можуть не надавати інформації про потужність своїх машин. У таких випадках використання централізованої системи планування є неможливим.

В **ієрархічній системі** планувальники організовані в ієрархію: планувальники вищого рівня оперують планувальниками нижчого рівня; планувальники найнижчого рівня оперують ресурсами. Основною перевагою ієрархічної системи планування є те, що вона дозволяє використання різних політик на різних планувальниках (а отже, у різних доменах). При цьому ієрархічна система має переваги централізованого управління: єдиний центральний планувальник виконує функцію мета-планування – розподіляє роботи на підпорядковані йому планувальники нижчого рівня відповідно до певного набору правил. Однак, у масштабних грід-системах, через велике навантаження на центральний планувальник, ієрархічна система планування може бути неефективною.

У **децентралізованій системі** (розподіленій системі) немає центрального планувальника – наявна множина рівноправних планувальників, кожному з яких підпорядкована певна група вузлів системи. Планувальники взаємодіють між собою, зокрема, можуть віддавати і приймати роботи від інших планувальників. Децентралізована система планування є добре масштабованою і стійкою до збоїв – вихід з ладу однієї частини системи не вплине на роботу інших частин. Тому таку систему можна застосовувати у величезних інтергрідах. Однак, децентралізоване планування може виконуватися неефективно, оскільки планувальники не “бачать” загальної картини у системі. Окрім цього, в децентралізований системі планування складно організувати паралельне виконання однієї роботи на кількох окремих доменах гріда.

Для вирішення цих проблем у масштабних грід-системах використовують розподілено-ієрархічну систему планування, яка володіє перевагами як ієрархічної так і децентралізованої системи. Детально архітектуру розподілено-ієрархічної системи розглянемо у наступному розділі.

Архітектура розподілено-ієрархічної системи управління ресурсами

Розподілено-ієрархічна система управління ресурсами грід-системи складається з локальних планувальників (ЛП) та зовнішніх планувальників (ЗП) (суперпланувальники або метапланувальники). ЗП керує декількома підпорядкованими йому ЛП. ЛП керує одним доменом за централізованим принципом.

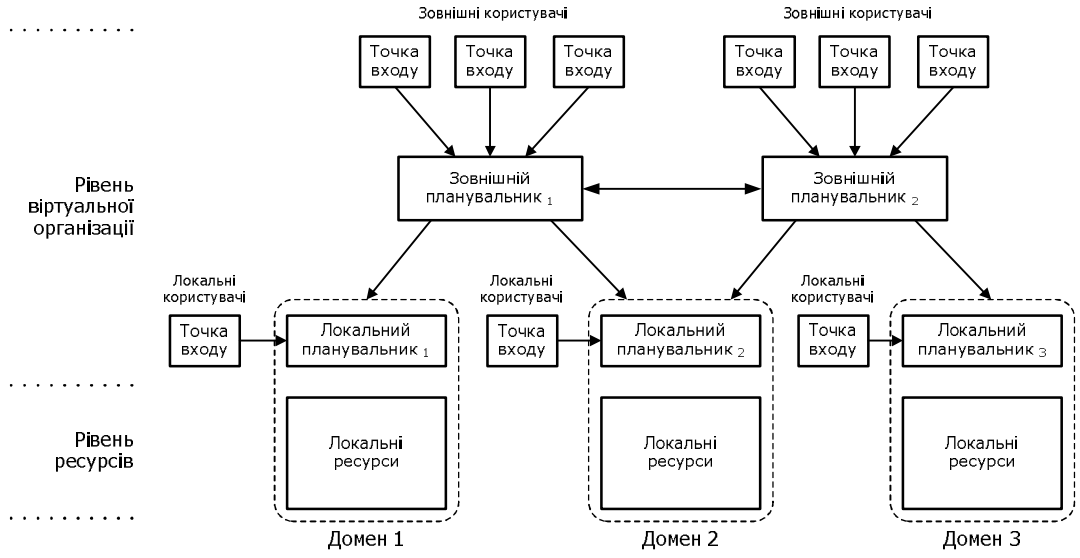

Рис. 1. Архітектура розподілено-ієрархічної системи управління ресурсами в грід

Користувачі гріда подають роботу в одну з точок входу. Деякі користувачі мають привілегії для безпосереднього доступу до ресурсів певних адміністративних доменів, тоді як решта користувачів працюють через ЗП. ЗП відповідає за розбиття роботи на менші роботи і розподіл їх по ЛП. Після отримання запиту на виконання роботи, ЗП опитує підпорядковані ЛП, щоб з’ясувати, які вузли можуть виконати роботу на своїх ресурсах і при тому вкластися у часові та бюджетні обмеження користувача (*перевірка виконуваності* [3]). Якщо ЗП не знаходить таких вузлів, він передає роботу до іншого ЗП.

Якщо кожен ЗП визначає роботу як невиконувану, то роботу віддають користувачу для перегляду поставлених вимог. Інакше, якщо придатний вузол знайдено, то ЗП передає роботу вибраному ЛП. ЛП відповідає за визначення порядку виконання робіт та виділення ресурсів для їх виконання. Для збільшення гнучкості системи ЛП, при отриманні нових важливих і термінових робіт, може віддати низькопріоритетні роботи ЗП для їхнього перерозподілу на інші ЛП [3].

Постановка задачі планування в грід-системі

Загальну роботу, яку потрібно виконати у грід-системі, подамо у вигляді напрямленого ациклічного графу робіт $G(V,E)$, де V – множина вершин $v_i \in V$, які представляють окремі неподільні роботи – набори інструкцій, які мають виконуватися на одній машині; E – множина комунікаційних ребер, визначає порядок слідування робіт; напрямлене ребро $e_{ij} = (v_i, v_j) \in E$ означає, що роботу v_j можна виконувати тільки після завершення та отримання результатів роботи v_i; роботу v_i називають батьківською, v_j – дочірньою. Бінарне відношення $v_i \,\mathbf{p}\, v_j$ означає, що роботу v_i потрібно виконати перед роботою v_j (при цьому v_i не обов'язково є безпосереднім батьком v_j). Вага $w(v_i)$ – специфікація роботи v_i (кількість обчислень), а вага $w(e_{ij})$ – специфікація комунікації ребра e_{ij} (кількість даних).

Функція $\mathbf{P}(v)$ визначає множину безпосередніх батьківських робіт роботи v; функція $\mathbf{C}(v)$ – множину безпосередніх дочірніх робіт. Роботу $v_{entry} = v \,|\, \mathbf{P}(v) = \varnothing$ називають *вхідною роботою* (початковою), а $v_{exit} = v \,|\, \mathbf{C}(v) = \varnothing$ – *вихідною роботою* (кінцевою). Якщо є більше однієї вхідної/вихідної роботи, то потрібно з'єднати їх єдиною вхідною/вихідною псевдороботою (роботою з нульовою кількістю обчислень та нульовими ребрами).

Топологію гетерогенної грід-системи подамо у вигляді ненапрямленого графу $S(P,L)$, де вершина $p_i \in P$ представляє i-ту обчислювальну машину, а ребро $l_{ij} = (p_i, p_j) \in L$ – двонапрямлене комунікаційне з'єднання обчислювальних машин p_i та p_j. Вага $w(p_i)$ визначає специфікацію обчислювальної машини p_i (відносну швидкодію), а вага $w(l_{ij})$ – специфікацію комунікаційного з'єднання l_{ij} (пропускну здатність). Будь-яка обчислювальна машина системи може одночасно виконувати обчислення та передавати дані.

Задачею планувальника є розподіл роботи, представленої графом робіт G по ресурсах системи S з врахуванням порядку слідування робіт. Розв'язком є план Ψ:

$$\Psi = G \to S = (\Phi, \mathrm{X}), \tag{1}$$

де Φ – *розподіл робіт*, множина значень f_i: i-та робота має виконуватися на f_i-й машині:

$$\Phi = \left\{ j_i : j_i \in \left[1, |P|\right] \,\middle|\, i = \overline{1, |V|} \right\}, \tag{2}$$

X – *послідовність виконання робіт*, множина унікальних значень c_i: спочатку виконується робота c_1, тоді c_2 і так до останньої роботи:

$$\mathrm{X} = \left\{ c_i : \left(c_i < c_j \forall v_{c_i} \,\mathbf{p}\, v_{c_j} \right) \wedge \left(c_i \neq c_j \forall i \neq j \right) \,\middle|\, i, j = \overline{1, |V|} \right\}. \tag{3}$$

Нехай функція $t(x,y)$ визначає: *а*) тривалість комунікації, якщо параметри – комунікаційне ребро e_{ij} та комунікаційне з'єднання l_{ij}; або *б*) тривалість виконання роботи, якщо параметри – робота

v_i та машина p_i; при тому, якщо роботу v_i не можливо виконати на машині p_i, то результатом буде ∞. Аналогічно, функція $c(x, y)$ визначає вартість комунікації або вартість виконання роботи.

Визначимо *час початку виконання* роботи v_i:

$$t_{start}(v_i) = \max\left[t_{DR}(v_i), t_{PR}(v_i)\right], \tag{4}$$

де $t_{DR}(v_i)$ – *час готовності даних* роботи v_i, тобто час завершення передачі результатів виконання всіх батьківських робіт $v_j \in \mathbf{P}(v_i)$ на машину p_{f_i}; $t_{PR}(v_i)$ – *час готовності машини* p_{f_i} до виконання роботи v_i, тобто час завершення виконання всіх попередніх робіт на цій машині.

Визначимо *час завершення виконання* роботи v_i – сума часу початку виконання та тривалості виконання роботи:

$$t_{finish}(v_i) = t_{start}(v_i) + t(v_i, p_{j_i}) \tag{5}$$

Часом завершення виконання задачі, представленої графом задач G, згідно з планом Ψ є час завершення виконання кінцевої роботи:

$$t(\Psi) = t_{finish}(v_{exit}) \tag{6}$$

Вартість виконання роботи, представленої графом задач G, згідно з планом Ψ є сума вартостей використання ресурсів:

$$c(\Psi) = \sum_{i=1}^{|V|} c(v_i, p_{j_i}) + \sum_{i=2}^{|V|} c(e_{i-1\,i}, l_{j_{i-1}\,j_i}) \tag{7}$$

Основними критеріями оптимізації при пошуку розв'язку (1) є:

1. Мінімізація загального часу виконання робіт в системі:

$$t(\Psi) \to \min \tag{8}$$

Якщо розв'язком (8) є ∞, то роботу, представлену графом задач G, неможливо виконати на системі S.

2. Мінімізація загальних затрат на виконання роботи в системі:

$$c(\Psi) \to \min \tag{9}$$

Для задоволення QoS-вимог планувальнику потрібно оптимізувати план за одним критерієм з врахуванням обмеження для інших критеріїв. Для цього потрібно розв'язати одну із часткових задач оптимізації, наприклад задачу часової або вартісної оптимізації:

1. Задача часової оптимізації – мінімізація часу виконання з врахуванням бюджетного обмеження:

$$\begin{cases} t(\Psi) \to \min, \\ c(\Psi) < B, \end{cases} \tag{10}$$

де B – максимальна фінансова вартість ресурсів, використаних для виконання роботи (бюджетне обмеження).

2. Задача вартісної оптимізації – мінімізація вартості виконання з врахуванням часового обмеження:

$$\begin{cases} c(\Psi) \to \min, \\ t(\Psi) < D, \end{cases} \tag{11}$$

де D – максимально допустимий час виконання роботи (крайній термін).

Задачі (8), (9), (10) та (11) називають *задачами відображення алгоритму на обчислювальну систему*. Знаходження найкращого розв'язку цих задач є NP-повною задачею [4]. Тому, використовують різноманітні евристичні алгоритми для пошуку наближених розв'язків.

Класифікація алгоритмів планування

Згідно з [5] сьогодні є два основні типи алгоритмів планування: на основі best-effort підходу (максимального використання можливостей) та на основі QoS-підходу (забезпечення якості обслуговування).

Планувальники на основі **best-effort підходу** спрямовані на досягнення певної конкретної мети: мінімізацію часу виконання робіт, максимізацію продуктивності системи, оптимізацію утилізації ресурсів тощо. При цьому планувальник не враховує грошової вартості використовуваних ресурсів і недостатньо піклується про забезпечення якості обслуговування інших користувачів. Планування робіт на основі best-effort підходу може використовуватися в так званих *партнерських грід* (community grid), в яких організації-учасники надають ресурси безкоштовно [5].

Користувачі грід можуть мати додаткові вимоги щодо виконання своїх програм – QoS обмеження. У [6] виділено п'ять QoS-вимірів: *часу*, *вартості*, *точності*, *надійності*, *безпеки*. Часове обмеження визначає максимальний час виконання робіт. Вартісне обмеження – максимальну вартість використання ресурсів. Обмеження точності визначає якісні характеристики результату виконання робіт. Обмеження надійності (або безвідмовності) стосується кількості збоїв під час виконання робіт. А обмеження безпеки вказує на рівень секретності робіт та надійність (в плані безпеки) ресурсів.

Планувальники на основі **QoS-підходу**, окрім задоволення функціональних вимог, намагаються адаптувати план під QoS-обмеження. Наприклад, стратегією планування може бути *часова оптимізація* – мінімізація часу виконання з врахуванням бюджетного обмеження; *вартісна оптимізація* – мінімізація вартості виконання з врахуванням часового обмеження (обмеження крайнього терміну).

Наближені розв'язки задач (8), (9) знаходять за допомогою алгоритмів на основі best-effort підходу, а задачі (10) та (11) розв'язують за допомогою алгоритмів на основі QoS підходу. Деякі алгоритми, розроблені на сьогодні, розглянемо у наступних розділах.

Алгоритми на основі best-effort підходу

Одним із найбільш досліджуваних алгоритмів планування є евристичний алгоритм **планування списком** (list-scheduling). Планування списком виконують у дві фази: 1) *визначення пріоритетів робіт*, на якій визначають послідовність виконання робіт та 2) *вибір ресурсів* – кожній роботі (згідно з порядком, визначеним в першій фазі) призначають машину, яка виконуватиме цю роботу. Планування списком називають *статичним плануванням списком*, якщо фаза вибору ресурсів починається після фази визначення пріоритетів робіт і називають *динамічним планування списком*, якщо ці дві фази виконують паралельно [7].

Прикладом статичного планування списком є один із найпростіших алгоритмів планування – HLFET (Highest Level First with Estimated Times). Для визначення пріоритетів робіт алгоритм визначає *статичний рівень* робіт.

1. Обчислити SL **(**статичний рівень**)** кожної роботи v_i **:**

$$SL(v_i) = w(v_i) + \max_{v_j \in \mathbf{P}(v_i)} SL(v_j)$$

2. Додати всі вершини в X в порядку спадання SL

3. Поки не кінець X **:**

призначити для роботи v_{c_i} машину**,** яка найшвидше виконає цю роботу**:**

$$j_i = \arg_{\mathrm{j}} \left[\min_{p_j \in P} t_{finish}(v_{c_i}, p_j) \right]$$

Рис. 2. Алгоритм статичного планування списком HLFET.

Прикладом динамічного планування списком є алгоритм ETF (Earliest Time First). На кожному кроці для всіх готових до виконання робіт визначають ту, яка буде виконаною найшвидше. *Готова до виконання робота* – робота, в якої всі батьківські вже виконані. Спірні ситуації вирішують вибором роботи з більшим статичним рівнем:

1. Обчислити SL **(**статичний рівень**)** кожної роботи v_i **:**

2. Додати до *списку готових до виконання робіт* вхідну роботу: $v_{entry} \to R$

3. Поки є готові до виконання роботи, $R \neq \varnothing$ **:**
Обчислити час виконання кожної роботи зі списку R на кожній машині і обрати пару робота-машина, яка дає найшвидший час завершення виконання роботи:

$$(c_i, j_i) = \arg \left[\min_{v_j \in R, p_k \in P} t_{finish}(v_j, p_k) \right]$$

Додати до списку R нові готові до виконання роботи: $\forall v \left| \mathbf{P}(v) \subset \mathrm{X} \to R \right.$

Рис. 3. Алгоритм динамічного планування списком ETF

Різні алгоритми планування списком по-різному визначають пріоритети робіт [7]. Окрім статичного рівня робіт використовують такі атрибути: t_{ASAP} (або EST – Earliest Start Time) – найраніший можливий час початку виконання роботи; t_{ALAP} (або LST – Latest Start Time) – найпізніший можливий час початку виконання роботи:

$$t_{ASAP}(v_i) = \begin{cases} 0, & якщо\ v_i = v_{entry} \\ \max\limits_{v_j \in \mathbf{P}(v_i)} \left[t_{ASAP}(v_j) + w(v_j) + w(l_{ji}) \right], & в\ іншому\ випадку \end{cases} \quad (12)$$

$$t_{ALAP}(v_i) = \begin{cases} t_{ASAP}(v_{exit}), & якщо\ v_i = v_{exit} \\ \min\limits_{v_j \in \mathbf{C}(v_i)} \left[t_{ALAP}(v_j) - w(l_{ji}) \right] - w(v_i), & в\ іншому\ випадку \end{cases} \quad (13)$$

У [7] розглянуто алгоритми HLFET та ETF для гомогенних систем – критерієм вибору машини в є *час початку* виконання роботи. В гетерогенних грід-системах машини мають різну обчислювану потужність, і машина, яка почне виконувати роботу найшвидше, не обов'язково найшвидше її виконає. Тому в описі алгоритмів як критерій вибору машини ми використали *час завершення* виконання роботи (див. також [8]).

Окрім евристичних алгоритмів планування у класі алгоритмів на основі best-effort підходу, є метаевристичні алгоритми. Метаевристичні алгоритми застосовують до дуже великих та складних задач – вони дають хороший розв'язок за невеликий час. Далі розглянемо основні метаевристичні алгоритми, які застосовують для планування у грід.

Алгоритм **GRASP** (Greedy Randomized Adaptive Search Procedure) – ітеративний метод пошуку плану випадковим чином [5]. На кожній ітерації алгоритм знаходить новий план і найкращий із знайдених планів є кінцевим планом. GRASP зупиняє своє виконання при задоволенні критерію зупинки, наприклад, при виконанні певної кількості ітерацій. Кожну ітерацію виконують у дві фази: 1) *побудови* та 2) *локального пошуку*.

На фазі побудови алгоритм генерує допустимий план (1), тобто план, у якому дотримано порядок слідування робіт та кожна робота є запланованою єдиний раз. Для вибору ресурсів алгоритм будує *обмежений список кандидатів* C (restricted candidate list, RCL), який містить ресурси-кандидати для виконання кожної роботи, і випадковим чином обирає ресурс із цього списку. RCL може містити k найкращих ресурсів (cardinality-based RCL) або всі ресурси, продуктивність яких перевищує деякий поріг (value-based RCL).

На фазі локального пошуку GRASP виконує покращення знайденого плану. Локальний пошук шукає локальні оптимуми в околі поточного розв'язку і генерує новий розв'язок. Якщо новий розв'язок є кращим за поточний (наприклад, має менший час виконання), то він замінює поточний.

Детальніше алгоритм GRASP та його використання у грід-системах розглянуто у [9].

Імітація гартування (Simulated Annealing) – метаевристичний алгоритм планування, розроблений на основі методу Монте-Карло [10]. Алгоритм ґрунтується на імітації фізичного процесу,

який відбувається при кристалізації речовини із рідкого стану в твердий (наприклад, при гартуванні металів). Процес проходить при поступовому зниженні температури. Вважають, що атоми вже вибудувалися в кристалічну решітку, але ще допустимі переходи окремих атомів із однієї комірки в іншу. Такий перехід відбувається з певною ймовірністю, при тому ймовірність зменшується зі зниженням температури. Атом або переходить в стан з меншим рівнем енергії або залишається на тому ж місці. Процес продовжується до утворення стійкої кристалічної решітки, яка відповідає мінімуму енергії атомів.

Поки не задоволено критерій зупинки:

1. Додати до списку готових до виконання робіт вхідну роботу: $v_{entry} \to R$

2. Поки є готові до виконання роботи, $R \neq \varnothing$:

Створити *обмежений список кандидатів* C для роботи c_i

Випадковим чином обрати машину з C для роботи c_i:

$$j_i = C_{\text{random}(|C|)}$$

Додати до списку R нові готові до виконання роботи: $\forall v \left| \mathbf{P}(v) \subset \mathrm{X} \to R \right.$

3. Виконати локальний пошук на знайденому плані Ψ:

$$\Psi' = localSearch(\Psi)$$

4. Обрати найкращий розв'язок:

$$\Psi_{best} = getBest(\Psi, \Psi', \Psi_{best})$$

Рис. 4. Алгоритм планування GRASP

На початку алгоритму генерують довільний план – машини для виконання робіт обирають випадково. Далі на кожній ітерації генерують новий розв'язок, випадковим чином змінюючи поточний план – одну з робіт переносять на іншу машину. Далі, за алгоритмом Метрополіса, залишають один із планів: порівнюють новий план з поточним і залишають новий план, якщо він є кращим. Кращим планом є той, за яким загальний час виконання робіт є меншим. Покращене значення часу позначають як db. В інших випадках новий план залишають з ймовірністю Больцмана $e^{\frac{-db}{T}}$, де T – поточна температура. Після проходження певної кількості ітерацій температуру зменшують. Алгоритм виконують, доки не буде досягнуто найнижчої дозволеної температури. Під час цього процесу алгоритм зберігає найкращий розв'язок і повертає його як оптимальний після завершення своєї роботи.

Генетичний алгоритм планування – метаевристичний алгоритм, який знаходить якісний розв'язок задачі планування і застосовує для цього принципи еволюції. Кожен розв'язок представлений особою (хромосомою). Генетичний алгоритм підтримує популяцію осіб, які найбільше еволюціонували. Якість осіб популяції визначають *функцією придатності* (fitness function). Значення цієї функції вказує, наскільки особа є кращою за інші особи популяції [5].

Типовий генетичний алгоритм виконують у такий спосіб. На початку створюють першу популяцію, яка складається з розв'язків, згенерованих випадковим чином. Далі, застосовуючи генетичні оператори селекції, схрещення (crossover) та мутації, утворюють нових нащадків. Визначають придатність кожної особи. Найбільш придатні особи використовують для створення наступної популяції. Алгоритм продовжують виконувати до задоволення критерію зупинки: до виконання певної кількості ітерацій або до досягнення певного рівня придатності.

Для створення генетичного алгоритму планування потрібно визначити спосіб представлення осіб, функцію придатності та генетичні оператори.

Найпоширенішим способом представлення осіб популяції є *двовимірне* представлення [5]. Наприклад, кожен план записують у вигляді двовимірного масиву: в одному вимірі вказують номери машин, а в іншому – роботи, які виконуватимуть на цих машинах. Для виконання генетичних маніпуляцій двовимірний масив конвертують в одновимірний [11]:

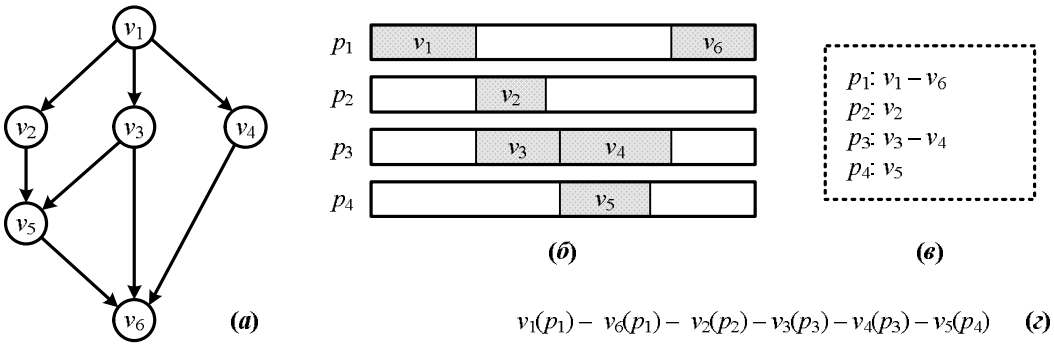

Рис. 5. Представлення плану (особи популяції) в генетичному алгоритмі планування: а) граф робіт; б) план виконання робіт; в) двовимірне та г) одновимірне представлення плану

Функція придатності повинна сприяти формуванню розв'язків, які найкраще задовольнятимуть мету планування. Наприклад, якщо метою є отримання мінімального часу виконання, то придатність особи можна обчислювати за формулою

$$F\left(\Psi_i\right) = \min_{\Psi_j \in \Upsilon} t\left(\Psi_j\right) - t\left(\Psi_i\right), \tag{14}$$

де Υ – поточна популяція. Особа з більшим значенням придатності є кращою.

Після обчислення придатності, нові особи порівнюють із особами попередньої популяції і виконують *селекцію* – залишають тих осіб, які найуспішніше еволюціонували у поточному поколінні. Різні методи селекції розглянуто у [5].

Інші два генетичні оператори – схрещення та мутації – застосовують для отримання нових осіб популяції. *Схрещення* виконують над двома особами популяції шляхом комбінування їх частин. Ідея схрещення полягає у тому, що комбінуванням двох придатних осіб можна отримати ще більш придатну особу. *Мутації* дають змогу дочірним особам отримати властивості, яких не мають батьківські і цим потенційно покращити генетичний матеріал. Частоту, з якою виконують мутації, визначають експериментальним коефіцієнтом. Детальніше спосіб виконання операцій схрещення та мутації у генетичному алгоритмі планування описано у [11].

Алгоритми на основі QoS підходу

На відміну від алгоритмів на основі best-effort підходу, алгоритми на основі QoS підходу при розв'язуванні оптимізаційної задачі враховують QoS-обмеження. Сьогодні підтримка QoS-обмежень в алгоритмах планування знаходиться на дуже попередньому етапі [5]. Більшість алгоритмів планування на основі QoS підходу виконують планування з часовим або бюджетним обмеженням. *Планування з бюджетним обмеженням* розв'язує задачу часової оптимізації (10), а *планування з часовим обмеженням* розв'язує задачу вартісної оптимізації (11).

Алгоритм **пошуку з поверненням** (Back-tracking), який розглянуто у [12] виконує планування з часовим обмеженням. На кожній ітерації алгоритму кожну готову до виконання роботу призначають на найдешевшу машину зі *списку наявних машин*. Складніші роботи призначають першими; серед машин однієї вартості обирають потужнішу. Після цього обчислюють загальний час виконання всіх призначених робіт. Якщо цей час перевищує часове обмеження, повертаються до попереднього кроку алгоритму, видаляють найдешевшу машину зі списку наявних машин цього кроку і перепризначають роботи. Якщо список наявних машин – пустий, то повертаються ще крок назад, звужують список наявних машин цього кроку і перепризначають роботи. Алгоритм виконують, доки для всіх робіт не будуть призначені машини.

Алгоритм **часового розподілу** (Deadline/Time Distribution), який описано у [13] – ще один евристичний алгоритм планування з часовим обмеженням. Алгоритм розділяє всі роботи на частини і кожній частині призначає кінцевий термін; відповідно, загальну задачу планування розділяє на підзадачі планування для кожної частини.

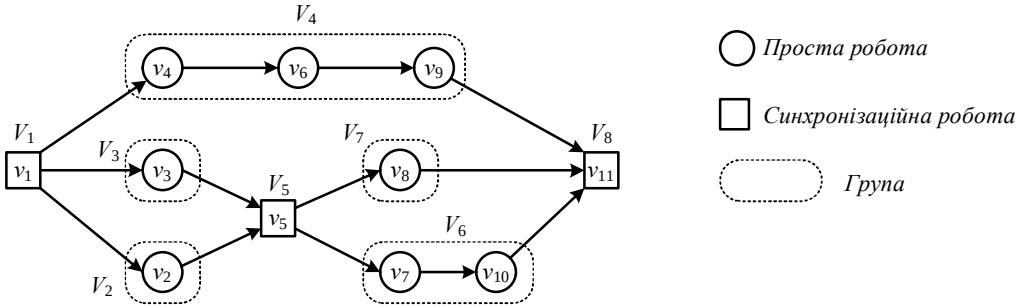

Рис. 6. Групування робіт

Всі роботи розділяють на *прості роботи* і *синхронізаційні роботи.* Синхронізаційна робота – робота, яка має більше однієї батьківської або дочірньої роботи. Інші роботи, які мають тільки одну батьківську і одну дочірню роботу, є простими. Прості роботи згруповують. Група – набір залежних простих робіт, які виконують послідовно між двома синхронізаційними роботами. Кожну синхронізаційну роботу теж виділяють у групу.

Після групування загальне часове обмеження ділять на частини, пропорційні мінімальному часу виконання групи, таким чином визначають кінцевий термін виконання кожної групи: $t_{deadline}^{V^i}$. Також для кожної групи додатково обчислюють час готовності $t_{ready}^{V^i}$ – найшвидший час, коли можна почати виконувати першу роботу та очікуваний час виконання $t_{estimate}^{V^i}$:

$$t_{ready}^{V^i} = \begin{cases} 0, & якщо\ v_{entry} \in V^i \\ \max\limits_{V^j \in \mathbf{P}(V^i)} t_{deadline}^{V^j}, & в\ іншому\ випадку \end{cases} \tag{15}$$

$$t_{estimate}^{V^i} = t_{deadline}^{V^i} - t_{ready}^{V^i} \tag{16}$$

Після цього обчислюють кінцевий термін виконання кожної роботи:

$$t_{deadline}^{v_i} = t_{ready}^{v_i} + t_{estimate}^{v_i} \tag{17}$$

$$t_{ready}^{v_i} = \begin{cases} 0, & якщо\ \mathbf{P}(v_i) = \varnothing \\ \max\limits_{v_k \in \mathbf{P}(v_i)} t_{deadline}^{v_k}, & в\ іншому\ випадку \end{cases} \tag{18}$$

$$t_{estimate}^{v_i} = \frac{\min\limits_{p_j \in P} t(v_i, p_j)}{\sum\limits_{v_k \in V^l} \min\limits_{p_m \in P} t(v_k, p_m)} t_{estimate}^{V^l}, v_i \in V^l \tag{19}$$

Маючи кінцевий термін виконання кожної роботи, можна скласти локальні оптимальні плани. Якщо кожен локальний план гарантує виконання своєї роботи в межах локального кінцевого терміну, то загальну роботу буде гарантовано виконано в межах загального кінцевого терміну; а вартісна оптимізація кожного локального плану приводить до вартісної оптимізації загального плану. Загальний план отримують об'єднанням всіх локальних планів.

У [11] описано **генетичний алгоритм** для планування з бюджетним та часовими обмеженням. Функція придатності складається з двох частин: *вартісної придатності* та *часової придатності*. Вартісну придатність обчислюють за формулою:

$$F_{cost}(\Psi_i) = \frac{c(\Psi_i)}{B^a \left[\max\limits_{\Psi_j \in \mathrm{Y}} c(\Psi_j)\right]^{1-a}} \tag{20}$$

де $a = 1$ якщо користувач задав бюджетне обмеження і $a = 0$ в іншому випадку.

Значення вартісної придатності сприяє 1) формуванню плану, який задовольнятиме бюджетне обмеження при плануванні з бюджетним обмеженням; 2) відбору планів меншої вартості при плануванні з часовим обмеженням.

Часову придатність обчислюють за формулою:

$$F_{time}\left(\Psi_i\right)=\frac{t\left(\Psi_i\right)}{D^b\left[\max\limits_{\Psi_j\in \mathrm{Y}} t\left(\Psi_j\right)\right]^{1-b}} \tag{21}$$

де $b=1$ якщо користувач задав часове обмеження і $b=0$ в іншому випадку.

Значення часової придатності сприяє 1) відбору планів з найшвидшим часом виконання при плануванні з бюджетним обмеженням; 2) формуванню планів, які задовольнятимуть часове обмеження при плануванні з часовим обмеженням.

Для планування з часовим обмеженням кінцева функція придатності має вигляд:

$$F\left(\Psi_i\right)=\begin{cases} F_{time}\left(\Psi_i\right), & \textit{якщо } F_{time}\left(\Psi_i\right)>1 \\ F_{\mathrm{cost}}\left(\Psi_i\right), & \textit{в іншому випадку} \end{cases} \tag{22}$$

Для планування з бюджетним обмеженням кінцева функція придатності має вигляд:

$$F\left(\Psi_i\right)=\begin{cases} F_{cost}\left(\Psi_i\right), & \textit{якщо } F_{\mathrm{cost}}\left(\Psi_i\right)>1 \\ F_{time}\left(\Psi_i\right), & \textit{в іншому випадку} \end{cases} \tag{23}$$

Висновки та майбутня робота

Евристичні алгоритми планування на основі best-effort підходу здатні знаходити хороший розв'язок задачі планування за поліноміальний час. Існує багато модифікацій евристичних алгоритмів, розроблених для планування робіт конкретного типу (специфічної структури, інтенсивних за обчисленнями та інтенсивними за даними, тощо).

Метаевристичні алгоритми розв'язують задачу планування ітеративно, використовуючи пошук випадковим чином. Перевагою метаевристичних алгоритмів є те, що вони знаходять оптимальний план, який ґрунтується на продуктивності виконання всіх робіт загалом, а не окремо взятих робіт, як у евристичних алгоритмах. Отже, на відміну від евристичних алгоритмів, метаевристичні можуть знаходити хороший розв'язок задачі планування для робіт різної специфіки, зокрема для робіт, представлених графом робіт великих розмірів та складної структури. Однак, час, необхідний для знаходження оптимального плану у метаевристичних алгоритмів, є значно більшим, ніж у евристичних.

Алгоритми планування робіт на основі best-effort підходу придатні для використання в невеликих інтрагрідах та партнерських грід, в яких учасники грід-системи надають ресурси безкоштовно, але не придатні для використання у корпоративних грід-системах та інтергрідах, які об'єднують декілька організацій і в яких важливим є забезпечення якості обслуговування. На відміну від алгоритмів планування на основі best-effort підходу, які враховують тільки одну цільову функцію (наприклад, оптимізацію часу), алгоритми планування на основі QoS-підходу виконують планування з врахуванням QoS-обмежень.

Більшість розроблених алгоритмів планування на основі QoS-підходу розглядають бюджетні або часові обмеження. Але у грід-системах, окрім обмежень бюджету та часу, користувачі можуть вимагати дотримання обмежень точності, надійності та безпеки. Окрім цього, у інтегрідах потрібно враховувати політику утилізації ресурсів кожного окремого домену. Тому актуальним є розробка алгоритмів планування, які будуть розв'язувати оптимізаційну задачу планування з багатовимірним QoS-обмеженням та кількома цільовими функціями, деякі з яких можуть конфліктувати.

1. Stockinger H. Defining the grid: a snapshot on the current view. // Springer Science+Business Media, LLC. / J Supercomput – 2007. – №42. – p.3-17. 2. Berstis V. Fundamentals of Grid Computing. //

IBM Corporation. – 2002. – 28p. 3. Grandinetti L. Operations Research Methods for Resource Management and Scheduling in a Computational Grid - a Survey / A. Attanasioa, G. Ghianib, L. Grandinettia, E. Guerrierob, F. Guerriero // Grid Computing: The New Frontier of High Performance Computing. – Elsevier Science and Technology Books, Inc. – 2005. 53-82 p. 4. Пономаренко В.С. Методы и модели планирования ресурсов в GRID-системах. / В.С. Пономаренко, С.В. Листровой, С.В. Минухин, С.В. Знахур // Монография. – Х.: ВД "Інжек". – 2008. – 408 с. 5. Yu J. Workflow Scheduling Algorithms for Grid Computing / J. Yu, R. Buyya, K. Ramamohanarao. // Springer-2008. – p.109-153. 6. Yu J. A Taxonomy of Workflow Management Systems for Grid Computing. J. Yu, R. Buyya // Grid Computing and Distributed Systems (GRIDS) Laboratory. – Department of Computer Science and Software Engineering. The University of Melbourne, Australia. – 2006. – 33p. 7. Hagras T. Static vs. Dynamic List-Scheduling Performance Comparison. / T. Hagras, J. Janeček. // Acta Polytechnica. – Czech Technical University Publishing House. – 2003. – Vol. 43. – No. 6. – p.16-21. 8. Sinnen O. Comparison of Contention Aware List Scheduling Heuristics for Cluster Computing. / O. Sinnen, L. Sousa. // Universidade Técnica de Lisboa. – 2001. – 6p. 9. Blythe J. Task Scheduling Strategies for Workflow-based Applications in Grids // IEEE International Symposium on Cluster Computing and the Grid. – 2005. 10. YarKhan A. Experiments with Scheduling Using Simulated Annealing in a Grid Environment. / A. YarKhan and J. J. Dongarra // The 3rd International Workshop on Grid Computing. – Baltimore, MD, USA. – 2002. 11. Yu J. Scheduling Scientific Workflow Applications with Deadline and Budget Constraints using Genetic Algorithms. / J. Yu, R. Buyya // Grid Computing and Distributed Systems (GRIDS) Laboratory. – Department of Computer Science and Software Engineering The University of Melbourne, VIC 3010 Australia. – 2005. – 22p. 12. Menascμe D. A. A Framework for Resource Allocation in Grid Computing / D. A. Menascμe, E. Casalicchio // The 12th Annual International Symposium on Modeling, Analysis,and Simulation of Computer and Telecommunications Systems (MASCOTS'04). - Volendam, The Netherlands. - 2004. 13. Yu J. A Cost-based Scheduling of Scientific Workflow Applications on Utility Grids / J. Yu, R. Buyya, and C.K. Tham // The First IEEE International Conference on e-Science and Grid Computing. – Melbourne, Australia. – 2005.