DOI: 10.34185/1991-7848.itmm.2026.01.057

МОДЕЛЮВАННЯ DAG-СТРУКТУР РОБОЧИХ НАВАНТАЖЕНЬ

Андрющенко В. О.[1] [ORCID], Танасієнко Д. О.[2] [ORCID]

[1]Український державний університет науки і технологій, к.т.н., доцент, Україна

[2]Український державний університет науки і технологій, аспірант, Україна

Анотація. *Запропоновано конструкційну модель породження DAG-структур робочих навантажень для тестування планувальників кластерів Kubernetes. Модель базується на апараті узагальненого конструктора та спеціалізує його компоненти для предметної області направлених ациклічних графів задач. Формалізовано механізм підстановки як частковий випадок контекстно-вільної графової граматики із заміною вузлів з чотирма типами правил: атрибутування, підстановка підграфа, реплікація та алгоритмічна генерація. Введено систему атрибутів, що розширює ідеї атрибутних граматик Кнута на графові структури. Доведено збереження ацикличності при кожній підстановці та завершимість процесу породження. Конструктор реалізовано мовою Python з декларативним YAML DSL. Експериментально підтверджено практичну придатність моделі на множині з 100 породжених DAG.*

Ключові слова: *інформаційні технології, програмне забезпечення, конструктивно-продукційне моделювання, направлений ациклічний граф, формальні граматики, графові граматики, планування задач, Kubernetes, оркестрація контейнерів, генерація робочих навантажень.*

Вступ. Тестування планувальників кластерів Kubernetes потребує реалістичних структур робочих навантажень — направлених ациклічних графів (DAG), що моделюють залежності між задачами Apache Airflow. Ручне створення DAG не масштабується, а випадкова генерація не відображає структурних патернів реальних конвеєрів (послідовні ланцюги, паралельні гілки, діамантові з'єднання). Існуючі параметричні генератори task graphs (DAGGEN [6], GGen [7]) не надають формальних гарантій структурних інваріантів породжених графів. Мета роботи — формалізувати породження DAG-структур засобами *конструктивно-продукційного моделювання* (КПМ) [3, 4], що забезпечує єдиний формалізм для опису процесів конструювання об'єктів. Наукова новизна полягає у застосуванні апарату узагальненого конструктора

до породження DAG-структур для реальної прикладної області з формальними гарантіями коректності (ациклічність та завершимість by construction).

Узагальнений конструктор та спеціалізація. Відповідно до [3], узагальнений конструктор визначається як трійка $C = \langle M, \Sigma, \Lambda \rangle$, де M — поповнений носій, Σ — сигнатура операцій, Λ — інформаційне забезпечення. Носій $M = T \cup F \cup K$ розділяється на термінальні елементи T, проміжні форми F та конструкції K. Процес конструювання є послідовністю застосувань правил підстановки: $f_0 \Rightarrow_{\psi_1} f_1 \Rightarrow_{\psi_2} \dots \Rightarrow_{\psi_n} k \in K$.

Спеціалізація $\mapsto_s : C \to C_{\text{DAG}}$ конкретизує компоненти трійки для предметної області — породження DAG задач:

$T_{\text{DAG}} = \{extract, transform, load, train, \dots\}$ — атомарні задачі;

$F_{\text{DAG}} = \{G = (V, E) | \exists\, v \in V : v \in N_{\text{DAG}}\}$ — графи з нетерміналами ($N_{\text{DAG}} = \{Sequential, Parallel, Diamond, \dots\}$);

$K_{\text{DAG}} = \{G = (V, E) | V \subset T_{\text{DAG}},\ G - \text{ациклічний}\}$ — завершені DAG.

Сигнатура $\Sigma_{\text{DAG}} = \langle \Theta_{\text{DAG}}, \Phi_{\text{DAG}}, R_{\text{DAG}} \rangle$ включає операції підстановки Θ_{DAG} (заміна нетерміналів на підграфи), атрибутування Φ_{DAG} (обчислення параметрів вершин) та відношення R_{DAG} — множину орієнтованих ребер (u, v) між вершинами графа, що задають залежності між задачами; при кожному застосуванні правила підстановки відношення оновлюються: додаються ребра E_{in} від попередників до вхідних вершин підграфа та E_{out} від вихідних вершин підграфа до наступників.

Механізм підстановки. Механізм підстановки є частковим випадком node replacement graph grammar [5] зі спеціалізованим правилом вбудовування, що гарантує збереження ацикличності DAG. Правило підстановки $\psi \in \Psi_{\text{DAG}}$ визначається як кортеж $\psi = \langle N, R_\psi, attach, g, guard \rangle$, де $N \in N_{\text{DAG}}$ — нетермінал, $R_\psi = (V_R, E_R)$ — ациклічний підграф-заміна, $attach = (sources, sinks)$ — вхідні та вихідні вершини підграфа, $g : V_R \to A_{intrinsic}$ — функція атрибутування, $guard : A \to \{true, false\}$ — предикат застосовності.

Операція $apply(G, m, \psi) = H$ будує результуючий граф: $V_H = (V_G \backslash \{v\}) \cup V_R$, $E_H = E_G^{-v} \cup E_R \cup E_{in} \cup E_{out}$, де $E_{in} = \{(u,s) | u \in pred_G(v), s \in sources(R_\psi)\}$, $E_{out} = \{(t,w) | t \in sinks(R_\psi), w \in succ_G(v)\}$.

Правила класифікуються на чотири типи: **I** (Attribute) — призначення атрибутів вершині без зміни структури графа; **II** (Substitute) — підстановка підграфа замість нетермінала; **III** (Replicate) — реплікація $v \to \{v_1, \dots, v_n\}$, де кількість реплік визначається атрибутом вершини; **IV** (Algorithm) — делегування спеціалізованому алгоритму генерації. Правило може містити предикат $guard$, що перевіряє наявність та значення атрибутів вершини; правило застосовується лише якщо guard виконується. Вибір правила здійснюється детерміновано за принципом *first-match*: вершини обходяться в топологічному порядку, для кожної перевіряються правила в порядку їх оголошення, і застосовується перше, guard якого виконується. Стохастичність структури породженого графа забезпечується каскадним механізмом: правила типу I призначають атрибути, значення яких семплюються з параметричних розподілів; guards правил типів II–IV залежать від цих атрибутів, тому варіативність значень атрибутів каскадно визначає структуру породженого графа.

Система атрибутів $A = A_{intrinsic} \cup A_{structural} \cup A_{scheduling}$ розширює ідеї атрибутних граматик Кнута [2] на графові структури: $A_{intrinsic}$ (cpu, memory, duration) призначаються при створенні; $A_{structural}$ (depth, width) обчислюються під час виведення та керують guards; $A_{scheduling}$ (earliest_start, slack, is_critical) обчислюються після повного породження.

Твердження 1 (Збереження ациклічності) *Якщо G — ациклічний граф, правило ψ коректне (R_ψ ациклічний, $sources \neq \varnothing$, $sinks \neq \varnothing$) і вбудовування здійснюється виключно через ребра від $pred_G(v)$ до $sources(R_\psi)$ та від $sinks(R_\psi)$ до $succ_G(v)$, то $H = apply(G, m, \psi)$ також ациклічний.*

Теорема 1 (Завершимість породження) *Нехай на скінченній множині типів нетерміналів N_{DAG} визначено строгий частковий порядок $\succ$ такий, що для кожного структурного правила $\psi = \langle N, R_\psi, \dots \rangle$ (типи II–IV) усі нетермінали у R_ψ мають типи строго нижчі за N: $\forall\, v \in V_R \cap N_{DAG}: N \succ type(v)$. Нехай також кожне*

правило атрибутування (тип I) має guard, що унеможливлює повторне застосування до тієї самої вершини за тим самим атрибутом. Тоді породження завершується за скінченну кількість кроків.

Ідея доведення. Кожне застосування структурного правила споживає один нетермінал типу N і може породжувати лише нетермінали типів $N' \prec N$. Хоча загальна кількість нетерміналів може тимчасово зростати (наприклад, при реплікації одного нетермінала на кілька), усі породжені нетермінали мають строго нижчий ранг. Оскільки $\succ$ — скінченний строгий порядок без нескінченних спадних ланцюгів, кількість застосувань структурних правил скінченна. Правила атрибутування (тип I) не змінюють множину нетерміналів; guard обмежує кількість їх застосувань до $|V| \times |A|$, де $|A|$ — кількість призначуваних атрибутів.

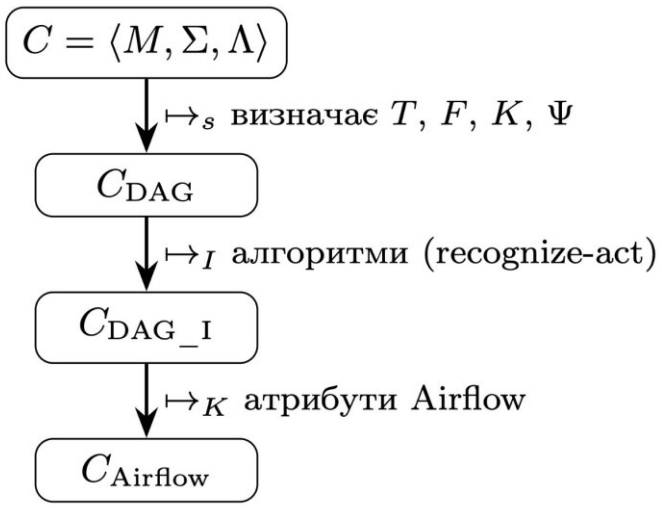

Рисунок 1 – Ланцюжок уточнень конструкційного компонента

Чисельний експеримент. Конструктор реалізовано мовою Python з декларативним YAML DSL для опису правил підстановки. Для підтвердження практичної придатності моделі породжено 100 DAG з 20–80 задачами, глибиною 3–7 та шириною до 12. Усі породжені графи задовольняють інваріанту ациклічності, а детермінованість забезпечується фіксацією seed генератора випадкових чисел.

Висновки. Запропоновано конструкційну модель породження DAG-структур на основі узагальненого конструктора [3]. Формалізовано механізм підстановки з чотирма типами правил, систему атрибутів та умовну стохастику.

Доведено збереження ациклічності та завершимості породження. На відміну від існуючих параметричних генераторів [6, 7], модель забезпечує формальні гарантії коректності by construction, генерацію структурно реалістичних DAG з контрольованими характеристиками та декларативний опис правил. Відкритим залишається аналіз розподілу ймовірностей на множині породжених конструкцій K_{DAG}.

ЛІТЕРАТУРА / REFERENCE

1. Chomsky N. Three models for the description of language // IRE Trans. on Information Theory. — 1956. — Vol. 2, No. 3. — P. 113–124.
2. Knuth D. E. Semantics of context-free languages // Mathematical Systems Theory. — 1968. — Vol. 2, No. 2. — P. 127–145.
3. Shynkarenko V. I., Ilman V. M. Constructive-Synthesizing Structures and Their Grammatical Interpretations. I. Generalized Formal Constructive-Synthesizing Structure // Cybernetics and Systems Analysis. — 2014. — Vol. 50, No. 5. — P. 655–662. DOI: 10.1007/s10559-014-9655-z.
.4. Skalozub V., Ilman V., Shynkarenko V. Ontological support formation for constructive-synthesizing modeling of information systems development processes // Eastern-European Journal of Enterprise Technologies. — 2018. — Vol. 5, No. 4(95). — P. 55–63. DOI: 10.15587/1729-4061.2018.143968.
5. Rozenberg G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformation. Vol. 1: Foundations. — World Scientific, 1997.
6. Cordeiro D., Mounié G., Perarnau S., Trystram D., Vincent J.-M., Wagner F. Random graph generation for scheduling simulations // Proc. 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010). — 2010. — P. 1–10. DOI: 10.4108/ICST.SIMUTOOLS2010.8667.
7. Canon L.-C., Sayah M., Héam P.-C. A Comparison of Random Task Graph Generation Methods for Scheduling Problems // Euro-Par 2019. LNCS, vol. 11725. — Springer, 2019. — P. 61–73.

CONSTRUCTIVE MODEL FOR GENERATING DAG STRUCTURES OF WORKLOADS

Vadym Andriushchenko, Dmytro Tanasiienko

Abstract. *A constructive model for generating DAG structures of workloads for testing Kubernetes cluster schedulers is proposed. The model is based on the generalized constructor formalism and specializes its components for the domain of directed acyclic task graphs. The substitution mechanism is formalized as a special case of node replacement graph grammar with four rule types: attribute assignment, subgraph*

substitution, replication, and algorithmic generation. An attribute system extending Knuth's attribute grammars to graph structures is introduced. Preservation of acyclicity under each substitution and termination of the generation process are formally proved. The constructor is implemented in Python with a declarative YAML DSL. Experimental validation on a set of 100 generated DAGs confirms the practical applicability of the model.

Keywords: *information technology, software engineering, constructive-synthesizing modeling, directed acyclic graph, formal grammars, graph grammars, task scheduling, Kubernetes, container orchestration, workload generation.*