УДК 683.519

СИМОНЕНКО В.П.,
КУРЕНЁВ А.С.

АЛГОРИТМ СТАТИЧЕСКОГО ПЛАНИРОВАНИЯ ДЛЯ GRID СИСТЕМ

В данной статье описан алгоритм статического планирования для многопроцессорных и Grid систем. Задача планирования сводится к поиску плана распределения задач по ресурсам с минимальным количеством вычислительных узлов с минимизацией времени решения. В этой статье предложен алгоритм пошагового конструирования, позволяющий уменьшить время планирования, за счет применения принципа одновременного прохода планировщика «сверху» и «снизу».

This article describes new static scheduling algorithm for multiprocessor and Grid systems. The main goal of scheduling is to find optimal distribution of tasks between resources with a minimum number of computing nodes while reducing processing time. In this article we propose step-by-step algorithm, which allows reducing scheduling time, by using the principle of simultaneous “forward” and “backward” scheduler’spassage.

1. Введение

В настоящее время ведутся активные исследования в создания инфраструктуры, которая обеспечивает надежный, устойчивый и недорогой доступ к высокопроизводительным вычислительным ресурсам. Такой инфраструктурой являются Grid-системы. Так же в данный момент в Украине эффективно развивается Украинский Академический Grid. Одной из актуальных задач в подобных системах является такое планирование заданий, которое позволяет наиболее эффективно использовать ресурсы вычислительной среды. В данном случае под планированием понимается распределение поступающих на вход заданий, подготовленных для параллельного выполнения на имеющиеся ресурсы. Целью планирования является определение эффективности распределения заданий при использовании различных критериев.

2. Обзор существующих решений

На данный момент разработано множество различных алгоритмов статического планирования [1], однако всем им присущ существенный недостаток, они требуют выполнять полное сканирование графа, причем зачастую несколько раз, что существенно замедляет работу самого планировщика. Большинство планировщиков для Grid систем, таких как, FCFS, SJF [2] просто выделяют определённое количество ресурсов под задачи, не выполняя оптимизацию внутри самих задач, что бы дало не только предпосылки к уменьшению времени самих выполнения задач, но и к более эффективному использованию ресурсов.[3]

3. Цель

Целью данной статьи является разработка нового алгоритма пошагового конструирования, позволяющего уменьшить время планирования, за счет применения принципа одновременного прохода планировщика «сверху» и «снизу».

4. Постановка задачи

Исходная задача поступает в систему в виде ориентированного ациклического графа (DAG) [4] задачи в ярусно-параллельной форме. Граф задается в виде множества:

$$G = \{N; E; W; C\},$$

где N – множество вершин графа; E – множество дуг (переходов между вершинами); W – веса (вычислительная сложность) вершин; C – веса (время коммуникации) дуг.

Для определения порядка запуска задач используется представление графа в виде матрицы связности: значение элемента матрицы указывает вес пути из одной вершины в другую, если такой существует. Эти значения также указывают зависимость одной задачи от другой.

При планировании необходимо учитывать условие предшествования. Задача готова к выполнению, когда все зависимости разрешены.

Для первой разрешённой задачи время запуска считается 0. Для остальных выбирается максимальное из всех определённых при разрешении зависимостей, таким образом, ко времени запуска задачи все её предшественники

будут выполнены, и время на пересылку данных учтено.

Таким образом, основной задачей планирования [5] является построение распределение (план) задач по доступным ресурсам так, чтобы минимизировать время выполнения приложения и использование ресурсов.

5. Описание алгоритма

На первом шаге алгоритма строится базовое решение. Под базовым решением понимается выделение отдельного процессора для каждой задачи. Такое решение используется в качестве основного, от которого производится поиск лучшего решения.

Основными атрибутами, по которым осуществляется планирование для DAG, являются t-level и b-level [1,6].

t-level – это самый длинный путь от начальной вершины до данной вершины n_i , без учёта веса n_i. Путь считается путём сложения всех весов вершин и пересылок, через которые он проходит. Этот параметр так же отражает наименьшее время начала вершины.T-level для i-й вершины вычисляется по следующей формуле:

$$tlevel\ (n_i) = max\ (tlevel(n_m) + w_m + c_{m,i}),$$

где$n_m = pred(n_i)$ – предшествующие вершины, w_m вычислительная сложность, $c_{m,i}$ – временные затраты на коммуникацию.

b-level- это самый длинный путь от вершины n_iдо исходной. Так же выделяют статический b-level, в котором не учитываются пересылки. B-level для i-го узла вычисляется по формуле:

$$blevel(n_i) = w_i + max\ (blevel(n_m) + c_{m,i}),$$

где $n_m = succ(n_i)$ – последующие вершины, w_i- вычислительная сложность, $c_{m,i}$ – временные затраты на коммуникацию. А статический b-level вычисляется по формуле:

$$sblevel(n_i) = w_i + max(sblevel(n_m)),$$

где $n_m = succ(n_i)$ – последующие вершины, w_i- вычислительная сложность.

Стоит отметить, что данные необходимо вычислять на каждом шаге алгоритма только для тех вершин, которые находятся на последующем уровне, при нисходящем планировании, и тех, которые находятся уровнем выше при восходящем. Это позволяет исключить повторное полное сканирование графа, что значительно ускоряет процесс планирования.

Обозначим множество вершин, принадлежащих k-му уровню:

$$n^k = \{n_i^k\}, i = \overline{1, m_k},$$

где n_i^k – i-я вершина k-го уровня, m_k – количество вершин на k-ом уровне.

Если вершина n_i^k имеет единственную инцидентную ей дугу e, и вершина n_j^{k-1} так же имеет единственную дугу e инцидентную ей, то вершины n_i^k и n_j^{k-1} необходимо кластеризовать.

При нисходящем планировании на каждом шаге алгоритма выполняются следующие действия:

- Вычисляется t-level для каждой вершины n_j^{k+1}, находящейся на следующем уровне.
- Рассматривается множество задач n^k находящихся на одном (k) уровне. Для каждой задачи k-го уровня выбирается множество задач $n_j^{k+1}, j = \overline{1, m_{k+1}}$, находящихся k+1 уровне, имеющих дугу, инцидентную рассматриваемой вершине n_i^k, т.е. вершина n_j^{k+1} имеет информационную зависимость от вершины N_i^k.
- Для кластеризации с N_i^k выбирается та вершина n_j^{k+1}, которая имеет наибольший t-level:max($tlevel\ (n_j^{k+1})$). Остальные вершины, имеющие информационную зависимость отрассматриваемой остаются без изменений, однако в последствии могут быть кластеризованы с другими вершинами k-го уровня.

При восходящем планировании на каждом шаге алгоритма выполняются следующие действия:

- Вычисляется b-level для каждой вершины n_j^{k-1}, находящейся на предыдущем уровне.
- Рассматривается множество задач n^k находящихся на одном (k) уровне. Для каждой задачи k-го уровня выбирается множество задач $n_j^{k-1}, j = \overline{1, m_{k+1}}$, находящихся k-1 уровне, имеющих дугу, инцидентную рассматриваемой вершине n_i^k, т.е. вершина N_i^k имеет информационную зависимость от вершины n_j^{k-1}.
- Для кластеризации с N_i^k выбирается та вершина n_j^{k+1}, которая имеет наибольший b-level: max($blevel\ (n_j^{k-1})$).

Для кластеризованных вершин, находящихся в одном узле, время затрачиваемое на коммуникацию приравниваем к 0.

Так же при планировании стоит учитывать вершины, обладающие свойствами неявной транзитности, т. е.для n_i^k вершины выполняется неравенство:

$$tlevel\left(n_i^k\right) \geq \min(tlevel\left(n_j^{k+1}\right)).$$

Такие вершины при планировании могут быть кластеризованы с вершинами, находящимися на более низком уровне, чем k+1. Зачастую это может привести к уменьшению ресурсов, требуемых для текущей задачи. Такую вершину целесообразно перенести на другой уровень для кластеризации, если выполняется соотношение:

$$ntlevel\left(n_i^k\right) \leq \max(tlevel\left(n_j^k\right)),$$

где $ntlevel$ –$tlevel$ для нового уровня.

Выполним планирование при помощи данного алгоритма для графа, представленного на рис. 1.

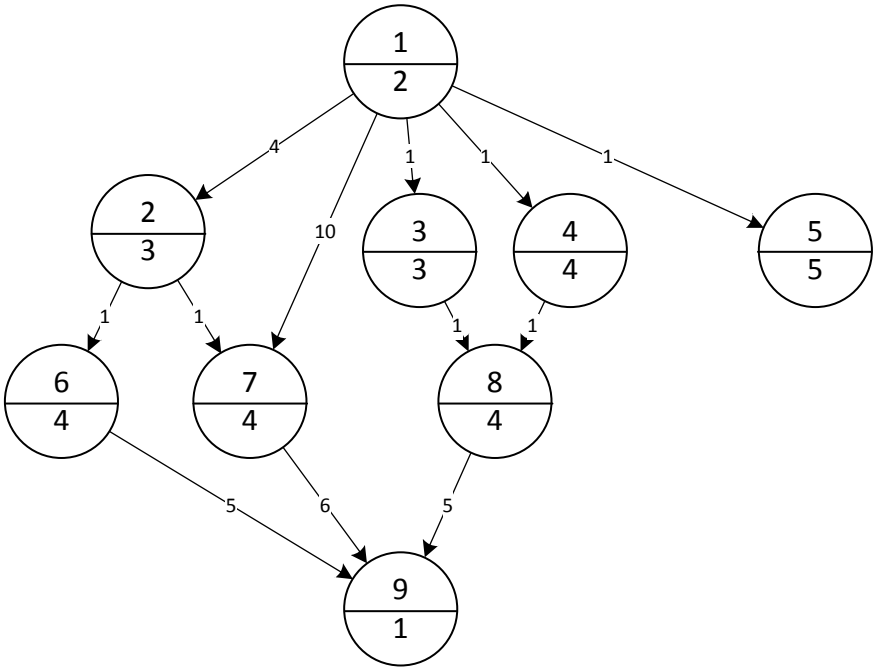

Рис.1. Исходный граф

Выполним построение базового решения, выделив каждой задаче по отдельному ресурсу R (рис. 2).

Распределим вершины по уровням:

$$n^1 = \{1\}$$
$$n^2 = \{2,3,4,5\}$$
$$n^3 = \{6,7,8\}$$
$$n^4 = \{9\}$$

Стоит отметить, что вершина 5 обладает свойством транзитности, поэтому она может находиться как на втором уровне, так и на третьем и четвертом.

Рассмотрим для вершины 1 (рис.3):

Из всех вершин второго уровня наибольший t-level имеет вершина 2:

$$tlevel(n_2) = 1 + 4 = 5$$

Поэтому вершину 2 кластеризуем с вершиной 1.

Теперь рассмотрим шаг для восходящего планирования (рис. 4).

Вершина 8, находящаяся на третьем уровне имеет наибольший b-level, поэтому она кластеризуется с 9 вершиной. Для вершин, которые находятся на одном ресурсе время, затрачиваемое на коммуникацию, приравниваем к нулю. В результате в начале следующего шага диаграмма будет выглядеть, как представлено на рис. 5.

Переходим на следующий уровень и для каждой вершины выполняем аналогичные действия. После выполнения шага 2 при нисходящем планировании получим диаграмму, представленную на рис. 6., в начале следующего шага диаграмма будет выглядеть, как представлено на рис. 5.

Переходим на следующий уровень и для каждой вершины выполняем аналогичные действия. После выполнения шага 2 при нисходящем планировании получим диаграмму, представленную на рис. 6.

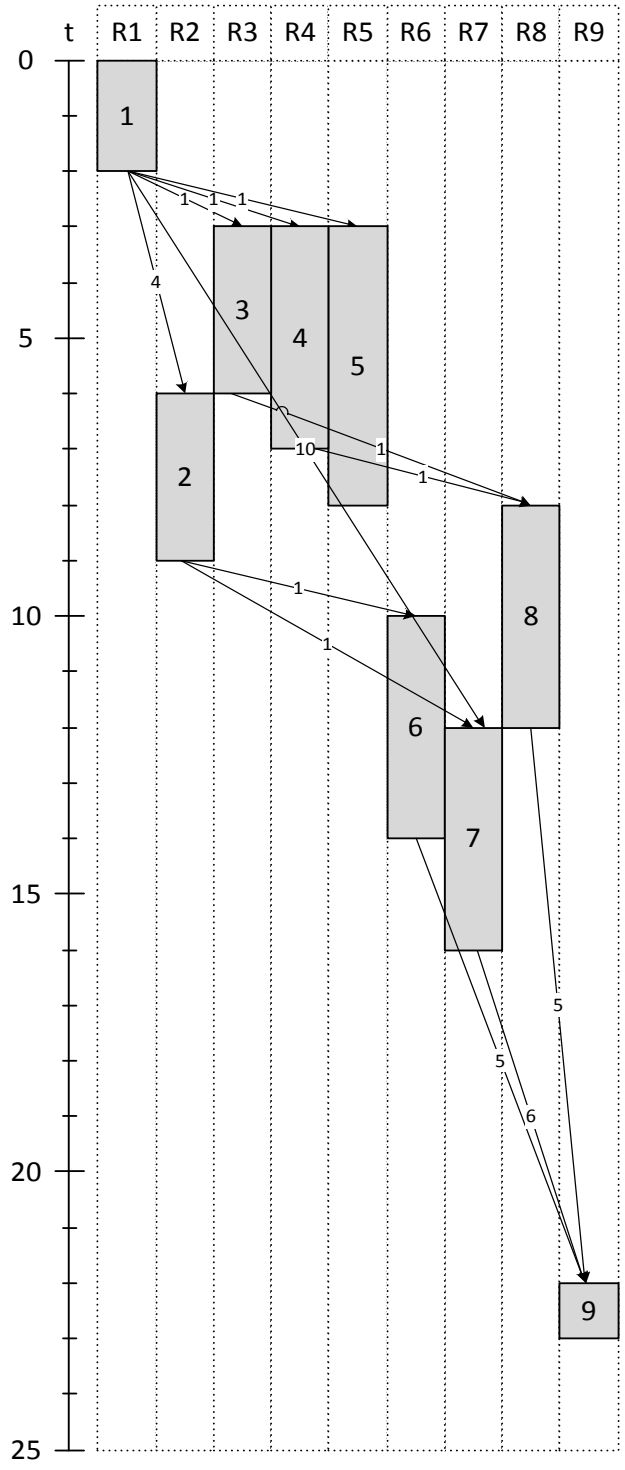

Рис. 2. Базовое решение

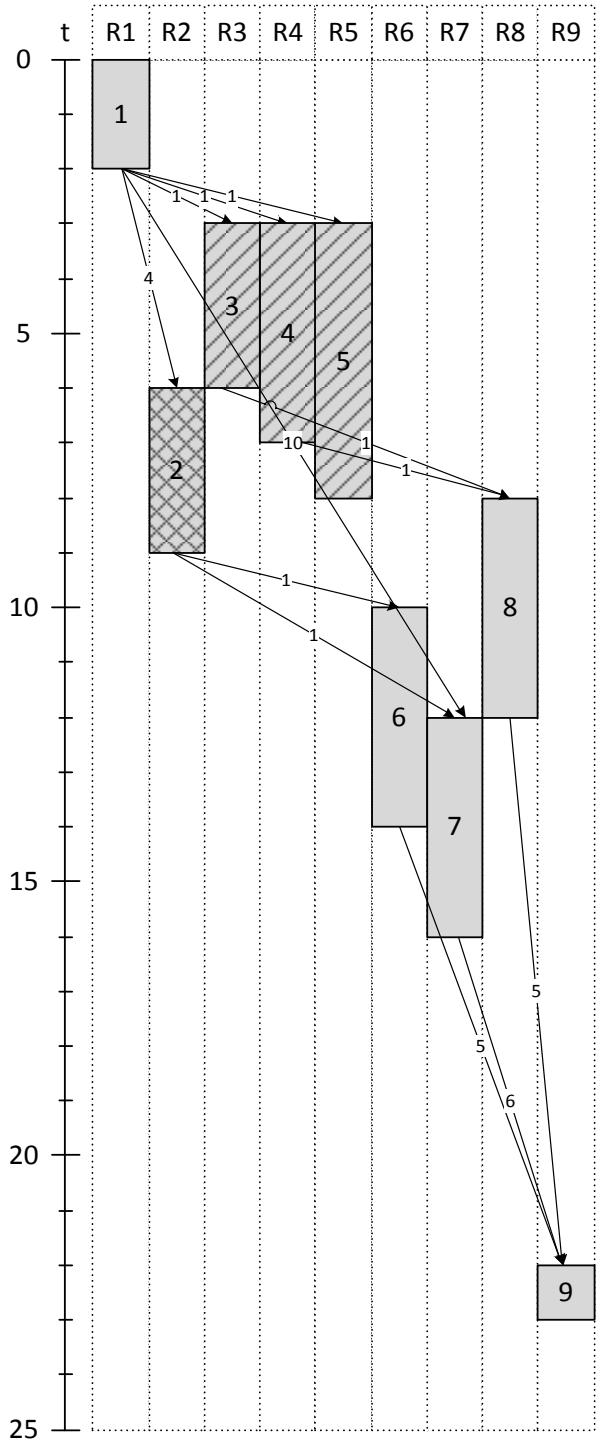

Рис. 3. Диаграмма Ганта. Шаг 1 (нисходящее планирование)

При выполнении шага для восходящего планирования берем во внимание свойство транзитности вершины 5. Т.к. Её можно кластеризовать с вершиной 1, то проверим, выполняется ли условие целесообразности данного действия:

$$ntlevel\ (n_5^4) \le tlevel\ (n_9^4),$$

$$ntlevel\ (n_5^4) = 9;\ tlevel\ (n_9^4) = 15,$$

Т.к. $9 < 15$, то вершину 5 стоит кластеризовать с вершиной 1.

Результирующая диаграмма Ганта представлена на рис. 7.

6. Результаты

В результате время выполнения, по сравнению с базовым решением сократилось на 30%. Для данного примера, описанный в данной статье алгоритм, показывает очень близкий к наилучшему результат, в сравнении с другими су-

ществующими алгоритмами [3], не только по критерию минимизации процессорного времени, затраченного навыполнения задачи, но и по количеству необходимых шагов (итераций) для осуществления планирования.

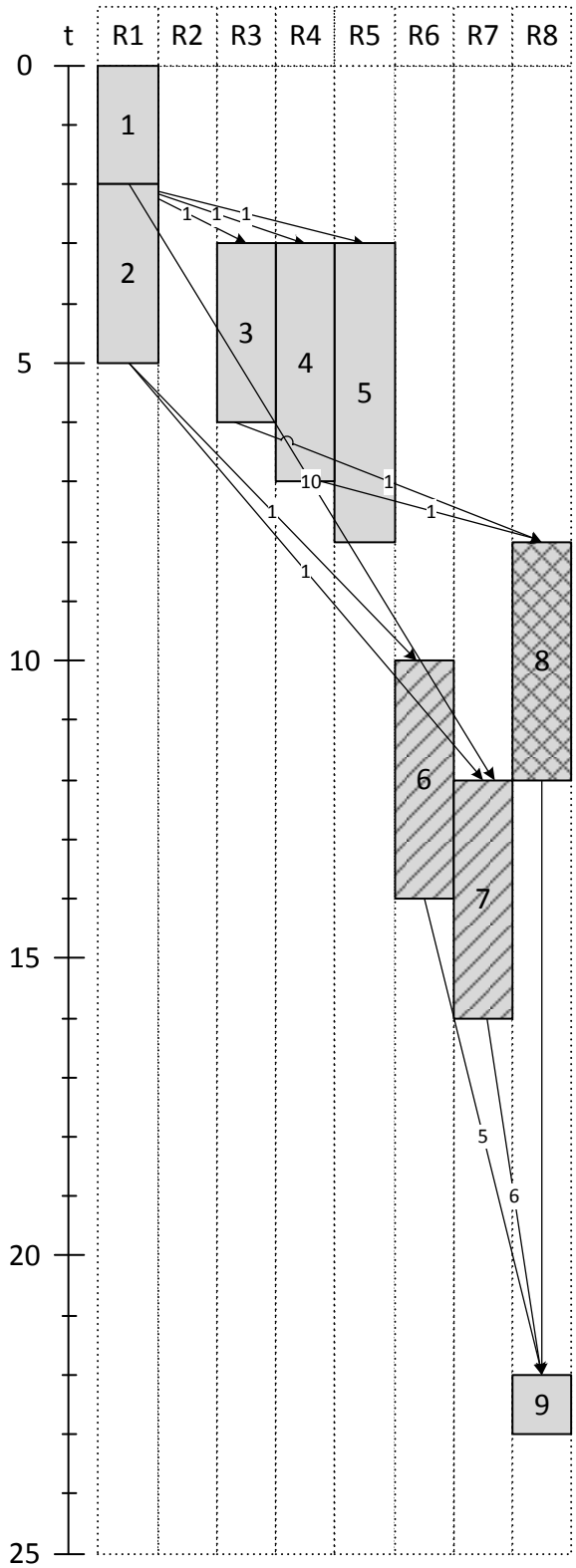

Рис. 4. Диаграмма Ганта. Шаг 1 с (восходящее планирование)

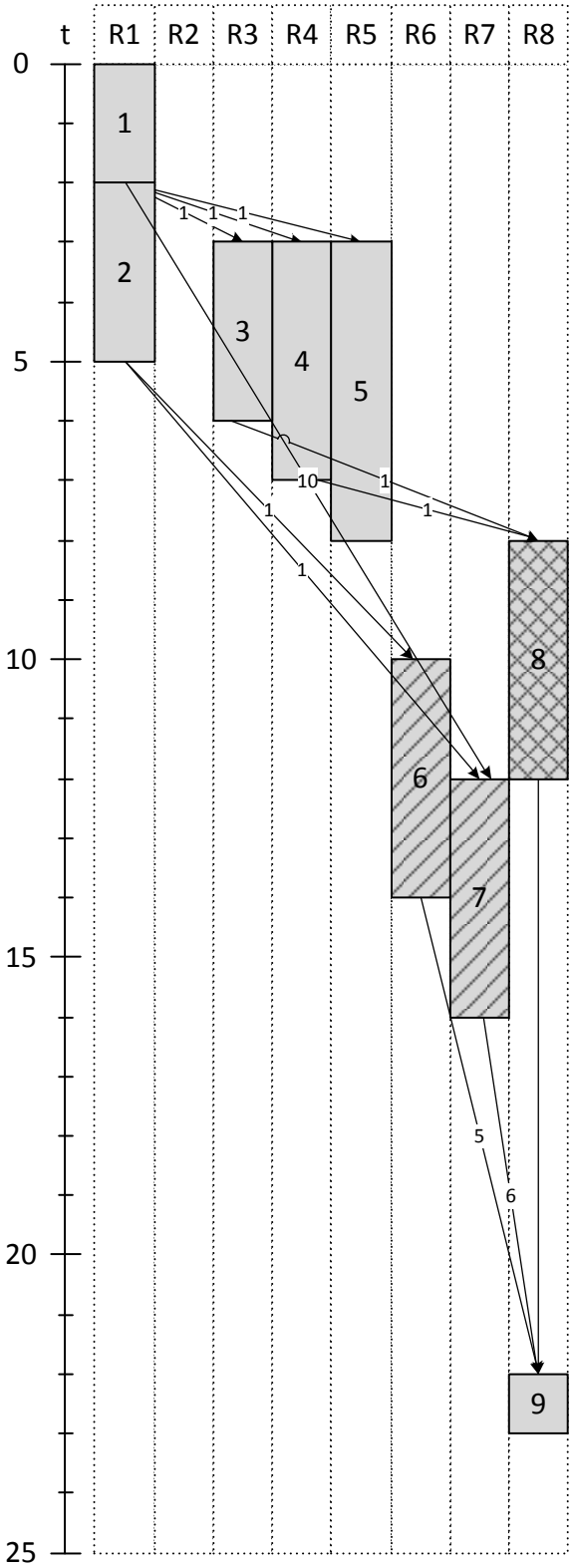

Рис. 5. Диаграмма Ганта. Конец шага 1

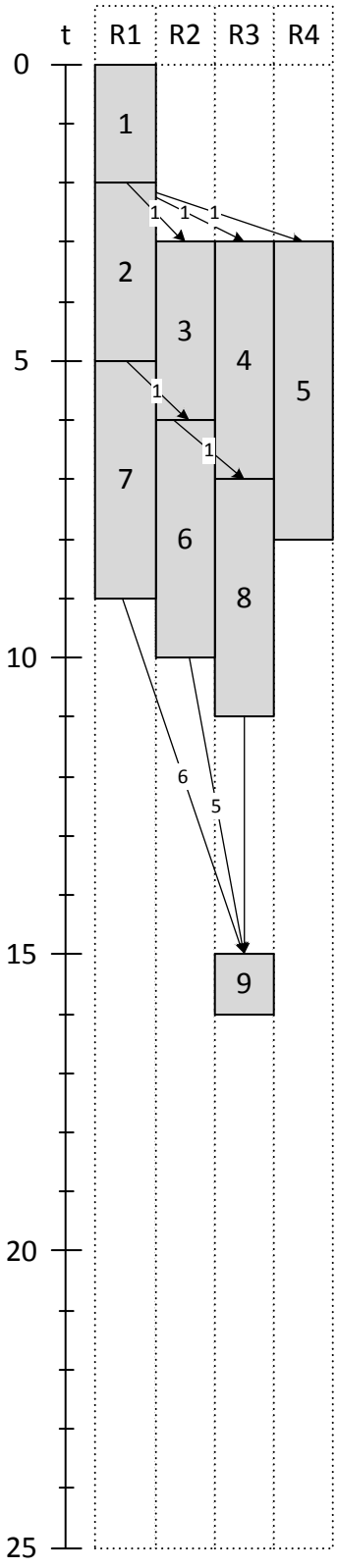

Рис. 6. Диаграмма Ганта. Шаг 2 (нисходящее планирование)

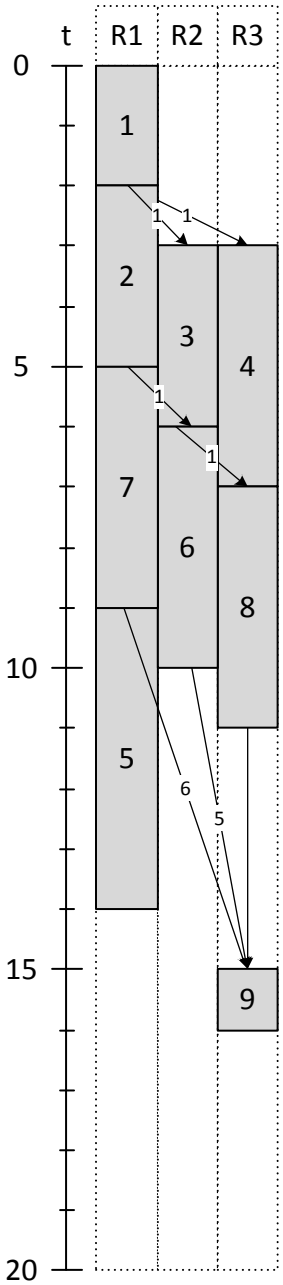

Рис. 7. Диаграмма Ганта. Результат

7. Дальнейшие перспективы

Для достижения лучших результатов планируется использовать в данном алгоритме методы, позволяющие оптимизировать планирование за счет дублирования вычислений на различных ресурсах, что даст предпосылки к уменьшению временных затрат на коммуникации, что в свою очередь может привести к сокращению времени выполнения всей задачи.

Список литературы

1. Y.-K. Kwok, I. Ahmad Static Scheduling Algorithms for Allocating Directed Task Graphs //ACM Computing Surveys. – 1999. – №4. – С. 406-471.
2. Коваленко В.Н. Управление параллельными заданиями в гриде с неотчуждаемыми ресурсами / В.Н. Коваленко, Е.И. Коваленко, Д.А. Корягин, Д.А Семячкин // Препринт No63. – М.: ИПМ РАН. – 2007. – С. 1-28.
3. М.А. Волк, Т.В. Филимончук, Р.Н. Гридель Методы распределения ресурсов для GRID-систем // ЗбірникнауковихпрацьХарківськогоуніверситетуПовітряних Сил. – 2009. – №19. – С. 100-104.
4. Ishfaq Ahmad and Min-You Wu, “Performance Comparison of Algorithms for Stat icScheduling of DAG to Multiprocessors”, [Електроннийресурс], режим доступу до журналу: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.42.8979&rep=rep1&type=pdf.
5. Shiyuan Jin, Guy Schiavone, DamlaTurgut , “A Performance Study of Multiprocessor Task Scheduling Algorithms” 43, C. 77-97, Jan 2008.
6. ParneetKaur, Dheerendra Singh, Gurvinder Singh &Navneet Singh, “Analysis, comparison and performance evaluation of BNP scheduling algorithms in parallel processing”. – International Journal of Information Technology and Knowledge Management // January-June 2011, Volume 4, No. 1, C. 279-284.