УДК 004.424

Сергей Григорьевич Рябчун

ТИПОВІ МАТЕМАТИЧНІ МОДЕЛІ НА ОСНОВІ МЕРЕЖ ПЕТРІ, ЩО ЗАСТОСОВУЮТЬСЯ ДЛЯ ФОРМАЛІЗАЦІЇ ПАРАЛЕЛЬНИХ ПРОЦЕСІВ В СИСТЕМІ МОНІТОРИНГУ ВИСОКОПРОДУКТИВНИХ КЛАСТЕРНИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМ

В ряду найбільш продуктивних суперкомп'ютерів більше 70% є високопродуктивні кластерні обчислювальні системи (далі КОС), що працюють під управлінням операційної системи Linux [1—3].

Досвід розробки і налагодження КОС показав, що одним з найбільш складних питань є введення в експлуатацію КОС, її поточний моніторинг і обслуговування. Основним інструментом системних адміністраторів при вирішенні цих питань є система моніторингу КОС. Нова КОС, що вводиться в експлуатацію звичайно є унікальною. При введенні КОС також розробляються нові системи моніторингу КОС, або вдосконалюються відомі. Це визначає ***актуальність даної роботи*** *і її практичну значимість.*

Попередні дослідження показують, що сьогодні для розробки програмного забезпечення (ПЗ) систем моніторингу КОС, в тому числі і військового призначення, доцільно використовувати мову програмування (МП) Erlang [3]. *Проте, як показує аналіз відомих джерел* [3—7], використання функціональної мови програмування (ФМП) Erlang призводить *до певних проблем* при розробці, тестуванні і супроводженні ПЗ, що пов'язані з певними особливостями цієї МП.

Першою особливістю є висока ємність коду Erlang. Другою особливістю є інша парадигма програмування, що відрізняється від парадигми розповсюджених в практиці імперативних мов програмування. Це призводить до ускладнень при розробці, тестуванні і спеціалізованої експертизі ПЗ. Знижується продуктивність розробки ПЗ. Згадані негативні явища викликані тим, що при розробці, тестуванні і супроводженні ПЗ використовуються інженерно-інтуїтивні методи. Причому використання інженерно-інтуїтивних методів ускладнює розробку ПЗ з великою кількістю паралельних процесів, що характерно для систем моніторингу КОС. Одним з шляхів усунення вищезгаданих проблем є використання математичного моделювання при розробці, тестуванні і супроводженні ПЗ з використанням ФМП.

Аналіз останніх досліджень і публікацій з ФМП Erlang [3—7] показав, що математичне моделювання при розробці ПЗ з використанням цієї МП не використовується. Всі матеріали щодо ФМП Erlang подаються на інженерно-інтуїтивному рівні, у вигляді опису фрагментів коду. В [8] подаються рекомендації з використання мереж Петрі для формалізації паралельних процесів для імперативної МП, що також не вирішує відкритого питання. Отже необхідна розробка математичних моделей (ММ) для формалізації процесу розробки, тестування і супроводження програмного забезпечення ФМП.

Таким чином, основною ***метою*** *даної роботи* є дослідження для побудови ММ для розробки ПЗ моніторингу КОС з використанням ФМП.

На першому етапі отриманні ММ на основі мереж Петрі основних конструкцій ФМП для послідовного програмування. На другому етапі отримані типові моделі на основі мереж Петрі для формалізації паралельних процесів в системі моніторингу КОС.

Зважаючи на вищевикладене, формальну постановку ***завдання*** *можна викласти в наступному виді: провести дослідження* типових конструкцій ПЗ для моніторингу КОС з використанням ФМП Erlang, визначити типові (ідіоматичні конструкції) МП, побудувати ММ типових конструкцій у вигляді мереж Петрі. В роботі акцент робить-

ся на моделюванні програмних засобів а не на теоретичну потужність мереж Петрі, отже буде використовуватися відомий математичний апарат мереж Петрі.

Формальний опис математичного апарату. Зважаючи на широкий спектр моделей мереж Петрі [9—11], введемо опис математичного апарату мереж Петрі для моделювання ФМП.

Графічно мережа Петрі представляється у вигляді дводольного орієнтованого графа з двома типами вершин — позиціями (їх в літературі іноді називають місцями) і переходами. Позиція позначається кругом, перехід рисою або прямокутником.

Рисою позначають простий перехід, прямокутником складний. В позиціях може бути розмітка у вигляді фішок [10, 11]. Наявність розмітки свідчить, що умова, яку формалізує позиція виконується. Наявність декількох фішок свідчить про те, що умова виконується з деяким запасом. Стосовно до ФМП розмічена позиція позначає наявність умов виконання, готовність до виконання, породження процесу при виконанні. Перехід звичайно використовується для позначення безпосередньо функції або фрагментів коду.

Перед викладом методики необхідно визначити обмеження на математичну модель у вигляді мережі Петрі. У методиці використається однорівнева, ординарна мережа Петрі і всі обмеження і формалізації що їй характерні [10, 11]. Кожному переходу ставиться у відповідність певна ієрархія, що відрізняє застосовану ММ від звичайних ординарних мереж Петрі. Ця ієрархія визначає порядок спрацьовування переходів при наявність декількох дозволених переходів.

Практика розробки і налагоджування ПЗ моніторингу КОС дає можливість представити типові конструкції МП у вигляді мереж Петрі. Прості конструкції МП внаслідок тривіальності реалізації розглядати не будемо. Побудуємо ММ найбільш характерних і нетривіальних конструкції ФМП.

До першої нетривіальної відноситься конструкція *if*. Кодова реалізація конструкції *if* показана на рис. 1 і працює в поданому далі порядку.

```
if
Guard_1 ->
        Action_1;
    Guard _2 ->
        Action_2;
    Guard _3 ->
        Action_3;
          ...
    Guard _n ->
        Action_n
end
```

Рис. 1. Конструкція if

Спочатку обчислюється контролер *Guard_ — Guard _n* (*n* належить множині натуральних чисел) , що показані на у фрагменті коду (рис. 1). Результати обчислень можуть приймати значення або *true* або *false*. Якщо *true*, то виконуються дії, які слідують за контролером, наприклад для *Guard _1* виконується *Action_1,* інші дії пропускаються до кінця конструкції. Якщо результат дії дорівнює *false,* обчислюється наступний контролер. Якщо жодна дія не повертає *true*, то буде видана помилка (виключення). Для запобігання видачі помилки наприкінці конструкції іноді додають атом *true*. Це дозволяє запобігти видачі помилки, навіть якщо в жодній попередній дії не буде результату *true*.

Зважаючи на логіку роботи конструкції *if* представимо її у вигляді мережі Петрі, як показано на рис. 2.

Для реалізації цієї конструкції доцільно використати перемикача Байера [10, 11], які можна представити у вигляді ординар-

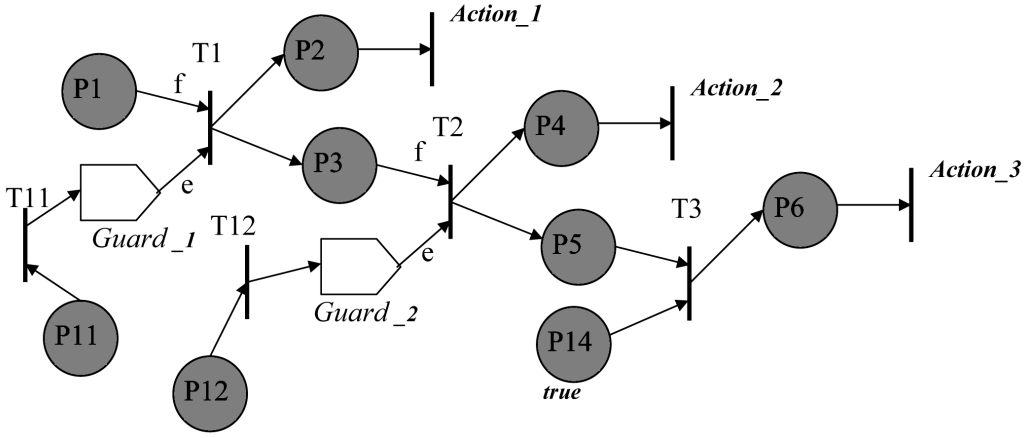

Рис. 2. Модель типової конструкції if

них мереж Петрі. Це дозволяє залишитися в рамках раніше введених обмежень на ММ. Перемикачі Байера дають можливість значно спростити ММ і представити її в узагальненому вигляді. Не змінюючи загального характеру досліджень представимо ММ конструкції *if*, яка містить дві дії і атом *true*. Місце P1 формалізує готовність конструкції до роботи. P11 і T11 (перехід може бути складним) формалізують виконання першої дії. Якщо дія виконується з результатом *true*, то в *Guard _1* розміщується мітка. При цьому спрацьовує перехід T1 і мітка розташовується в P2.

Виконується послідовність *Action_1* і подальші дії не виконуються. Якщо дія виконується з результатом *false*, мітка розташовується в P3, що формалізує готовність до наступної дії і перевірки її результату. Аналогічна послідовність здійснюється для переходу T2. У випадку, якщо жодного разу не було результату *true*, здійснюється перехід до T3 і мітка розташовується в P6 і виконується відповідна послідовність дій. Збільшуючи кількість переходів типу T1, T2, T3 можна необмежено збільшувати кількість альтернатив в конструкції рис. 1.

Представимо ММ конструкції *case*. Її узагальнений програмний код показаний на рис. 3, а докладний опис роботи приведено в [4, 5]. Коротко роботу *case* можна представити у наступній послідовності: обчислюється *expression,* результати обчислень порівнюється з шаблоном *Pattern_ i.* Якщо вони співпадають, то виконуються відповідні дії, що слідують за шаблоном і подальша перевірка не виконується. Якщо порівняння показало негативний результат, здійснюється перевірка з наступним шаблоном. У випадку відсутності жодного збігу повертається {*error, unknown_element*}.

Зважаючи на логіку роботи конструкції *case*, представимо її у вигляді мережі Петрі, як показано на рис. 4. Відповідно рис. 4 модель *case* працює аналогічно ММ *if* і для її реалізації використовуються перемикачі Байера.

```
case expression of
Pattern_1 -> expressi-
on_1, expression_2, …;
Pattern_2 -> expressi-
on_1, expression_2, …;
Pattern_3 -> expressi-
on_1, expression_2, …;
        …
Pattern_n -> expressi-
on_1, expression_2
      end
```

Рис. 3. Конструкция case

Місце P1 формалізує готовність переходу до обчислень. Перехід T1 формалізує порівняння з шаблоном. Позитивний результат при наявності мітки в *Pattern_1*, негативний результат відсутність мітки. Якщо результат порівняння *true* (наявність мітки в *Pattern_1*), спрацьовує перехід T1 і мітка розміщується в P2, якщо результат порівняння негативний, мітка розміщується в P3, що символізує перехід до іншого порівняння. Якщо жодна умова не виконується (P5) спрацьовує перехід T3 і повертається {error, unknown_element}. Місце P6 символізує готовність до подальшого виконання програми. Отже ММ на рис. 2 і рис. 4 аналогічні одна одній, незначна відмінність полягає в деяких елементах. Так же немає труднощів зробити ММ у вигляді мережі Петрі будь якої послідовної структури ФМП (try ... catch, рекурсивні і інші конструкції).

Наступним етапом є розробка типових ММ паралельних структур ФМП з виростанням мереж Петрі при розробці системи моніторингу КОС. Практика розробки і налагоджування програм моніторингу КОС дає можливість представити типові паралельні структури і конструкції МП у вигляді мереж Петрі. Семантика паралельного програмування і наявність додаткових процесів відрізняють ММ послідовного програмування. Відмінність полягає в постійної наявності в пам'яті КОС інформації про процес, можливості прийняття ним повідомлень і пере-

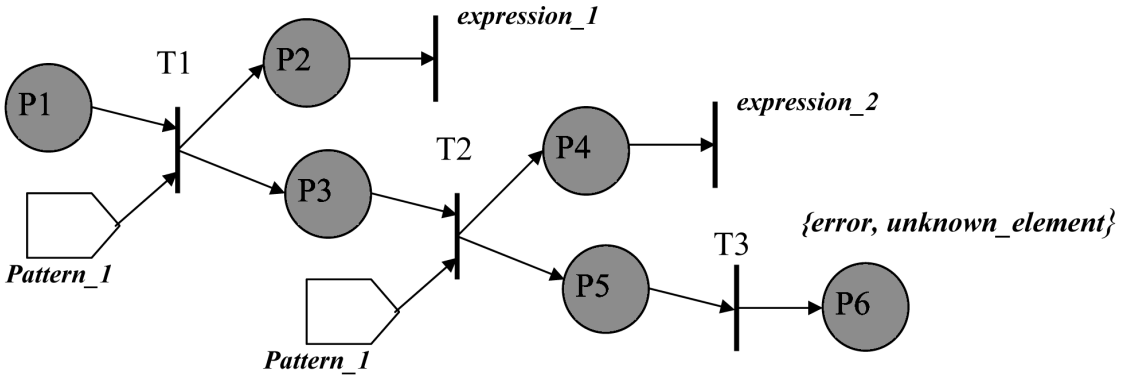

Рис. 4. Модель типової конструкції Case

дачу повідомлень. Крім того ММ повинна визначати ідентифікацію і статус процесу.

Для породження паралельних процесів використовується вбудована функція *spawn (module1, proc1, [arg1, arg2, arg3])* [4, 5]. Породження нового процесу можна представити у вигляді ММ, що показана на рис. 5.

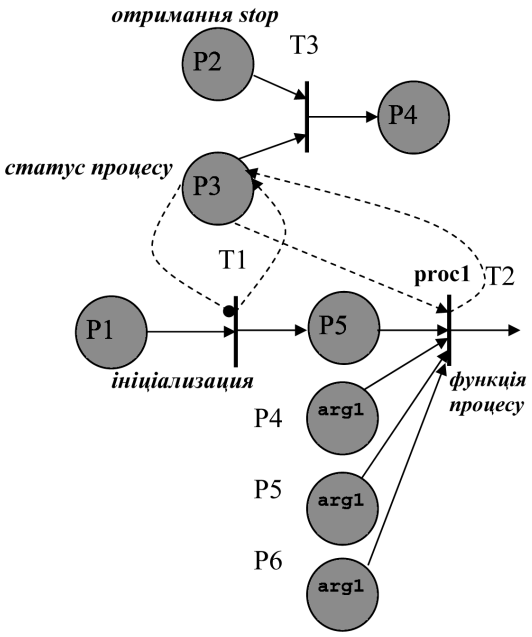

Рис. 5. Типовий варіант моделі життєвого циклу процесу

Місце P1 формалізує готовність до ініціалізації нового процесу. Перехід T1 формалізує ініціалізацію процесу. При ініціалізації процесу в P3 розташовується мітка, яка формалізує його статус. Наявність мітки означає що процес існує, відсутність мітки означає факт знищення процесу. Місце P5 формалізує готовність до виконання основної функції процесу. Інгібіторна стрілка на перехід T1 формалізує неможливість повторної ініціалізації процесу з однаковим ідентифікатором. Перехід T2 формалізує виконання основної функції процесу. Місця P4—P6 формалізують наявність вхідних даних для виконання функції. Ребро, що позначено пунктиром, формалізує можливість виконання основної функції процесу тільки при наявності у нього статусу "живий" . Перехід T3 формалізує надходження сигналу знищення процесу. В цьому випадку в P2 розташовується мітка. Переходу T3 встановлюється найбільший пріоритет для конструкції. При надходженні мітки в P2 мітка з P3 видаляється і процес набуває статусу "не живий".

Іншою важливою конструкцією в паралельному програмуванні є конструкція очкування процесом повідомлення від іншого процесу — *receive*. В цій роботі розкриті два найбільш характерних варіанта варіанти її застосування. На рис. 6 показаний фрагмент коду першої реалізації конструкції. Дослідження показали, що її можна представити у вигляді ММ, що показана на рис. 7. На ньому, не змінюючи загального характеру досліджень, показано два альтернативних варіанту прийому повідомлення. Приклад програмного коду такої реалізації показаний на рис. 6. Перехід T1 (рис. 7) формалізує підготовку до виконання роботи конструкції. Місце P3 показує готовність до альтернативного прийому повідомлень, місткість P3 примусово встановлюється рівним 1. Місця P6 і P7 формалізують надходження повідомлення *foo* або *bar* (рис. 6). При надходженні відповідного повідомлення в P7 або P6 розташовується мітка. Місце P9 або P8 формалізують надходження додаткових даних при реалізації функції.

```
receive
foo-> Action_11;
      Action_12
end
receive
bar->
      Action_21;
      Action_22
end
```

Рис. 6. Код першого варіанту використання receive

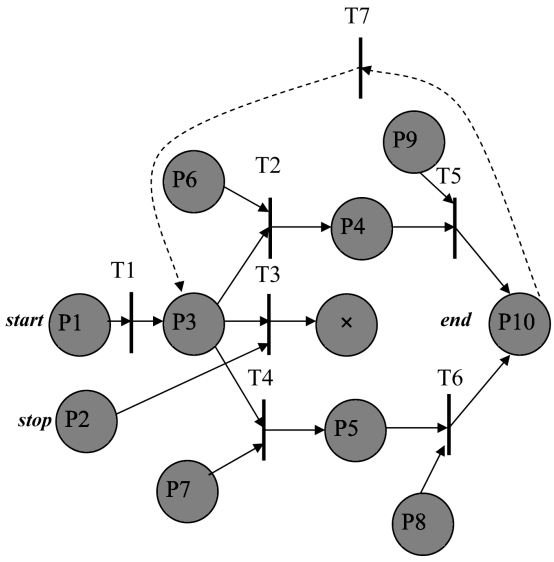

Рис. 7. Типова модель конструкції receive (варіант 1)

Робота моделі. При переході програми до цієї конструкції в P1 розташовується мітка, спрацьовує перехід T1 мітка розташовується в P3 і ММ готова до роботи. Для організації альтернативного вибору в ММ є

переходи T2 і T4. Зважаючи на місткість P3, яка дорівнює одиниці, спрацювати може тільки один з переходів T2 і T4. Спрацює саме той перехід, на який скоріше прийде повідомлення. Це буде формалізовано у вигляді наявності мітки в одному з місць P7 або P6.

Приведемо приклад. Нехай в готову до роботи ММ прийде повідомлення *foo* (рис. 6). В місце P6 розташовується мітка і спрацьовує перехід T2, формалізуючи готовність до виконання *Action_11 і Action_12*. Перехід T5 формалізує ці функції або дії. Місце P9 формалізує надходження зовнішніх даних для виконання *Action_11 і Action_12*.

При надходженні повідомлення *bar* конструкція працює аналогічно. На відміну від *foo,* при цьому виконуються альтернативні переходи T4, T6 і реалізуються інші функції або дії. Таким чином здійснюється двохальтернативний вибір. Місце P10 формалізує закінчення виконання роботи конструкції. Ребро що показано пунктиром дає можливість формалізувати повторення роботи конструкції декілька разів у вигляді циклу. Місце P1 і перехід T1 формалізує примусовий вихід з циклу. Представлена ММ (рис. 7) реалізує вибір між двома повідомленнями. Це не змінює узагальнюючого характеру ММ. Додаючи додаткових переходів виду T2, T4 можна формалізувати потрібну кількість альтернатив в ММ.

На рис. 8 показаний фрагмент коду другої реалізації конструкції *receive*. На відміну програмного коду, що показаний на рис. 6, ця конструкція приймає будь яке повідомлення і реалізує однакову для всіх повідомлень послідовність дії.

```
receive
Msg-> Action_11;
      Action_12
     Action_21;
       Action_22
end
```

Рис. 8. Код другого варіанту використання receive

Дослідження показали, що типову конструкцію у вигляді фрагменту коду (рис. 8) можна представити у вигляді ММ, що показана на рис. 9. На рис. 9, не змінюючи загального характеру досліджень, показаний прийом повідомлень від двох джерел. Цей процес формалізують P6, T2 і P7, T4 відповідно. Модель аналогічна рис. 7. Відрізняється лише тим, що при находженні першого з повідомлень від одного з джерел, виконується одна для всіх повідомлень послідовність дій, яку формалізує перехід T5.

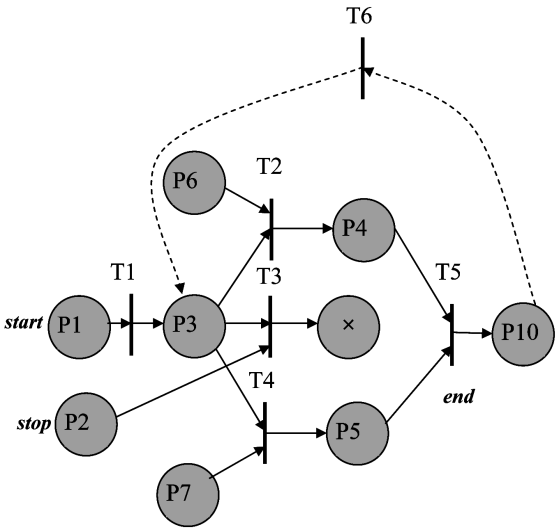

Рис. 9. Типова модель конструкції receive (варіант 2)

Для перевірки адекватності ММ здійснювався запуск модуля в якому містились типові конструкції, для яких були побудовані ММ. Звірка роботи конструкцій (типових функцій) в модулі ФМП і робота відповідної мережі Петрі показала, що ММ, побудовані з використанням розробленої методики є адекватними вихідному коду ФМП.

Висновки за результатами статті. В результаті проведених досліджень були побудовані типові ММ на основі мереж Петрі для формалізації послідовних і паралельних процесів для системи моніторингу КОС. Висока ємність коду Erlang і характерна парадигма програмування дає можливість використати ці моделі для розробки ПЗ з використанням ФМП. Використання цих моделей для імперативних мов програмування сумнівно, оскілки в них низка ємність коду, що призведе до значного збільшення і ускладнення ММ, зменшення її наочності. Крім того наявність класів в більшості імперативних МП потребує додаткових досліджень для використання мереж Петрі для моделювання ПЗ. На відміну від імперативних МП, у ФМП використання ММ у вигляді мереж Петрі доцільно внаслідок високої ємності коду. Як показала практика розробки і впровадження систем моніторингу КОС, використання ММ у вигляді мереж Петрі дозволяє: спростити розробку, тестування і супроводження ПЗ з використанням ФМП; спростити експертизу ПЗ; збільшити продуктивність розробки ПЗ, особливо для фрагментів коду, в яких багато паралельних процесів, що характерно для системи моніторингу КОС; використовувати при розробці ПЗ математичні методи аналізу.

В якості напрямків подальших досліджень необхідно розробити узагальнені ММ для серверної і перешкодозахищеної конструкції ФМП.

Література

1. Воеводин В. В. Параллельные вычисления / В. В. Воеводин, Вл. В. Воеводин. — СПб. : БХВ-Петербург, 2002. — 608 с. **2. Лацис А.** Как построить и использовать суперкомпютер / А. Лацис — М. : Бестеллер, 2003. — 240 с. **3. Трескин М.** Инструменты интроспекции в Erlang OTP / М. Трескин. — Практика функционального программирования. — 2010. — № 5. Режим доступу: www.fprog.ru/2010. **4. Baruch R.** A guide to functional programming in Erlang for the experienced procedural developer / R. Baruch. — San Francisco : Creative Commons, 2007. — 31 p. **5. Armstrong J.** Concurrency Oriented Programming in Erlang / Joe Armstrong. — Distributed Systems Laboratory Swedish Institute of Computer Science, 2003. — 12 p. Mode of access: www.guug.de/veranstaltungen/ffg2003/papers/ffg2003-armstrong.pdf. — title from the screen. **6. Armstrong J.** Programming Erlang: Software for a Concurrent World / J. Armstrong. — N. Y. : Pragmatic Programmers, 2007. — 536 p. **7. Cesarini F.** Erlang programming / F. Cesarini, S. Thomson. O'Reilly Media. — N. Y. : Sebastopol, 2009. — 470 p.

8. Федотов И. Е. Некоторые приемы параллельного программирования / И. Е. Федотов. — М. : Москов. гос. ин-т радиотехники, электроники и автоматики, 2008. — 188 с. **9. Murata T.** Petri Nets: Properties, Analysis, and Applications / T. Murata. Proceedings of the IEEE, 1989. — Vol. 77. — № 4. — P. 541—580. 10. Питерсон Дж. Теория сетей Петри и моделирование систем / Дж. Питерсон. — М. : Мир, 1984. — 264 с. 11. Котов В. Е. Сети Петри / В. Е. Котов. — М. : Наука, 1984. — 160 с.

В статье решаются две задачи по исследованию процесса разработки программного обеспечения мониторинга кластерной вычислительной системы (КВС). При решении первой задачи получены математические модели на основе сетей Петри основных конструкций функционального языка программирования для последовательных конструкций системы мониторинга КВС. В ходе решения второй задачи получены типовые математические модели на основе сетей Петри для формализации параллельных процессов в системе мониторинга КВС. Полученные решения дают возможность улучшить продуктивность разработки программного обеспечения.

Ключевые слова: математические модели, сети Петри, кластерная вычислительная система, функциональный язык программирования, программное обеспечение.

Article describes solution problems during research of software engineering for system monitoring high-performance clusters (HPC).

At first Petri net mathematical models build for sequential principal components of system monitoring HPS on the basis of functional programming language. Secondly Petri net mathematical models build for concurrent principal components and parallel processes of system monitoring HPS. The received decisions give the chance to improve efficiency of software engineering for systems monitoring HPS.

Key words: mathematical model, Petri net, high-performance clusters (HPC), applicative language, software.