ҚАЗАҚСТАН РЕСПУБЛИКАСЫ ҒЫЛЫМ ЖӘНЕ ЖОҒАРЫ БІЛІМ МИНИСТРЛІГІ

«Л.Н. ГУМИЛЕВ АТЫНДАҒЫ ЕУРАЗИЯ ҰЛТТЫҚ УНИВЕРСИТЕТІ» КЕАҚ

Студенттер мен жас ғалымдардың
«ǴYLYM JÁNE BILIM - 2023»
XVIII Халықаралық ғылыми конференциясының
БАЯНДАМАЛАР ЖИНАҒЫ

СБОРНИК МАТЕРИАЛОВ
XVIII Международной научной конференции
студентов и молодых ученых
«ǴYLYM JÁNE BILIM - 2023»

PROCEEDINGS
of the XVIII International Scientific Conference
for students and young scholars
«ǴYLYM JÁNE BILIM - 2023»

2023
Астана

УДК 001+37
ББК 72+74
G99

«ǴYLYM JÁNE BILIM – 2023» студенттер мен жас ғалымдардың XVIII Халықаралық ғылыми конференциясы = XVIII Международная научная конференция студентов и молодых ученых «ǴYLYM JÁNE BILIM – 2023» = The XVIII International Scientific Conference for students and young scholars «ǴYLYM JÁNE BILIM – 2023». – Астана: – 6865 б. - қазақша, орысша, ағылшынша.

ISBN 978-601-337-871-8

Жинаққа студенттердің, магистранттардың, докоранттардың және жас ғалымдардың жаратылыстану-техникалық және гуманитарлық ғылымдардың өзекті мәселелері бойынша баяндамалары енгізілген.

The proceedings are the papers of students, undergraduates, doctoral students and young researchers on topical issues of natural and technical sciences and humanities.

В сборник вошли доклады студентов, магистрантов, докторантов и молодых ученых по актуальным вопросам естественно-технических и гуманитарных наук.

УДК 001+37
ББК 72+74

ISBN 978-601-337-871-8

Пайдаланылған әдебиеттер тізімі

1. ISO/IEC 27000: Information technology – Security techniques – Information security management systems – Overview and vocabulary (2016)
2. IBM Corporation: IT Security Compliance Management Design Guide with IBM Tivoli Security Information and Event Manager. 2nd edn. (2010). http://www.redbooks.ibm.com/abstracts/sg247530.html?Open.
3. Techtarget: Security information and event management (SIEM) (2014). http://searchsecurity.techtarget.com/definition/security-information-and-event-management-SIEM.
4. Scarfone, K.: Introduction to SIEM services and products (2015). http://searchsecurity.techtarget.com/feature/Introduction-to-SIEM-services-and-products.
5. Miller, D., Harris, S., Harper, A., VanDyke, S.: Security Information and Event Management (SIEM) Implementation. McGraw-Hill, New York (2010). 464 p.
6. Miloslavskaya, N.G., Senatorov, M.Y., Tolstoy, A.I.: Information Security Incident and Business Continuity Management. Information Security Management Issues Series, 2nd edn., vol.3,170 p. Goriachaja linia-Telecom, Moscow (2014).
7. Verizon: Data Breach Investigations Report (2015). http://www.verizonenterprise.com/DBIR/2015/.

УДК: 004

ИССЛЕДОВАНИЕ МЕТОДОВ КЭШИРОВАНИЯ ПРИ УПРАВЛЕНИИ ДАННЫМИ

Марат Марта Мараткызы
магистрант 2 курса специальности «Информационные системы» maratkyzy.m@gmail.com,
научный руководитель, доктор философии (PhD), и.о.доцент кафедры Информационные системы
Касекеева А.Б.
ЕНУ им Л.Н. Гумилева, г. Астана

Аннотация

В этой работе рассмотрены известные алгоритмы кэширования. Была построена аналитическая модель системы кэширования. Также был предложен и реализован алгоритм, особенностью которого является учет нагрузки на сеть необходимой для повторной передачи данных при их вытеснении из кэша. Были проведены сравнения эффективности алгоритмов на различных тестовых наборах.

Введение

Первой технологией, которая использовалась с целью снижения времени ожидания ответов от сервера на запросы пользователей и снижения трафика, было кэширование. Постоянное развитие сетевых технологий и глобальной сети Интернет обуславливают рост числа исследований во всем мире направленных на повышение производительности систем, основанных на данных технологиях. Одним из основных направлений исследований является использование кэширования как главного инструмента для снижения нагрузки на систему в целом, увеличения ее производительности. Обычно кэширование определяется как хранение ответов на запросы с целью их последующего повторного использования[1]. Кэширование преследует следующие цели:

- сократить время ожидания ответа пользователем;
- уменьшить нагрузку на сервер;
- уменьшить нагрузку на сеть.

Аналитическая модель кэширования

Аналитическая модель системы кэширования позволяет понять структуру системы и связи между ее отдельными элементами. С ее помощью могут быть оптимально определены параметры настройки системы для достижения максимальной эффективности.

Было замечено, что распределение относительной частоты запросов соответствует закону Зипфа [2], который гласит, что если все запросы упорядочить по убыванию частоты их

использования, то относительная вероятность i-го запроса приблизительно обратно пропорциональна его номеру.

Глассман [3] обнаружил, что данные запрашиваются из Интернета с относительной частотой, которая соответствует певому закону Зипфа:

$$\frac{\Omega}{i}, \qquad (1)$$

где Ω - нормализовочная константа, а i - номер запроса, в списке всех запросов, упорядоченных в порядке убывания частоты их использования.

Дальнейшие исследования [4] показали, что распределение относительной вероятности запросов в случае определенной группы пользователей соответствует обобщенному закону Зипфа:

$$\frac{\Omega}{i^{\propto}}, \qquad (2)$$

где $0<\alpha<1$.

Пусть количество различных запросов пользователей равно S и все запросы упорядочены в порядке убывания их частоты. Тогда вероятность i-го запроса $\mathrm{P}_S(i)$ будет вычисляться по формуле:

$$\mathrm{P}_S(i) = \frac{\Omega}{i^{\propto}}. \qquad (3)$$

Из условия нормировки

$\sum_{i=1}^{S} \mathrm{P}_S(i) = 1$ (4)

следует, что

$$\Omega = \left(\sum_{i=1}^{S} \frac{1}{i^{\propto}}\right)^{-1}. \qquad (5)$$

Обозначим общее количество запросов полученных системой кэширования как N. Разделим множество всех запросов пользователей на два подмножества: «краткосрочной полезности» и «долгосрочной полезности». Данные, которые запрашивались только один раз, принадлежат первому множеству. Данные, которые запрашивались более одного раза, принадлежат второму множеству. Обозначим мощность первого множества как N1, а мощность второго множества как N2.

Максимальная эффективность H_{max} для идеальной системы с неограниченным кэшем вычисляется как отношение числа повторных запросов к общему количеству запросов полученных системой:

$$\mathrm{H}_{max} = \frac{N - S}{N}. \quad (6)$$

В реальных системах объем кэша ограничен, поэтому при повторном запросе запрашиваемые данные могут уже покинуть кэш. Эффективность реальных систем кэширования H_{real} во многом зависит от алгоритма замещения. На основании сравнения идеальной и реальной производительности можно ввести коэффициент эффективности, который будет показывать насколько наша система близка к идеальной:

$$\mathrm{E} = \frac{\mathrm{H}_{real}}{H_{max}}. \quad (7)$$

В качестве фундаментальных законов, которые характерезуют поведение системы кэширования, рассмотрим обобщенный закон Зипфа(3) и нормировочное соотношение (4) [5].

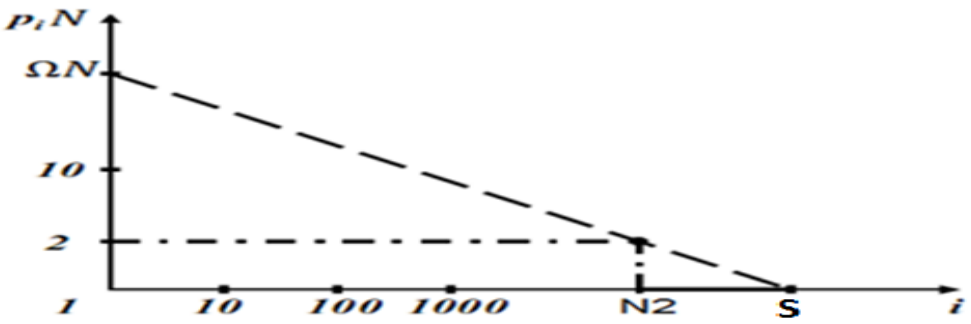

Рисунок 1. Особые точки распределения Зипфа.

Каждое из этих отношений можно применить к любой точке графического представления обобщенного закона Зипфа. Используем точки (N1, 1), (N2,2) для построения аналитической модели [5]. Подставим (S, 1), (N2,2) в (3), применим для (N2,2) нормировочное условие и получим систему уравнений, описывающих поведение системы кэширования:

$$\frac{\Omega N}{N_2{}^{\propto}} = 2, \quad (8)$$

$$\frac{\Omega N}{S^{\propto}} = 1, \quad (9)$$

$$\int_1^{N_2} \frac{\Omega}{x^{\propto}} dx = H_{max}. \quad (10)$$

Уравнение (10) описывает поведение идеальных систем кэширования с неограниченным размером кэша. Для реальных систем с размером кэша C записей оно примет вид:

$$\int_1^{C} \frac{\Omega}{x^{\propto}} dx = H. \quad (11)$$

В частности последнее уравнение можно использовать для определения влияния изменения размера кэша на эффективность системы кэширования. Пусть размер кэша равен C_1. Если мы хотим установить размер кэша равным C_2 , то прогнозируемое изменение эффективности системы можно вычислить следующим образом:

$$H_1 = \int_1^{C_1} \frac{\Omega}{x^{\propto}} dx \quad (12)$$

$$H_2 = \int_1^{C_2} \frac{\Omega}{x^{\propto}} dx \quad (13)$$

$$\frac{H_2}{H_1} = \left(\frac{C_2}{C_1}\right)^{1-\propto} \quad (14)$$

Для системы уравнений (8), (10) может быть получено уравнение для экспериментального нахождения α:

$$\alpha = 1 - \frac{2N_2}{HN}. \quad (15)$$

Рассмотренная модель описывает ситуацию, когда запрашиваемые данные являются статическими, то есть не обновляются со временем. Внесем в нее необходимые изменения для получения динамической модели, как это было предложено в [6]. Пусть количество пользователей системы равно N. Общее количество различных запросов пользователей обозначим как n, а среднее количество запросов пользователя – λ. Время между последовательными обновлениями данных распределяется экспоненциально с параметром μ. Тогда максимальная эффективность системы определяется как

$$C_N = \int_1^n \frac{1}{Cx^{\propto}} \frac{\lambda N \frac{1}{Cx^{\propto}}}{\lambda N \frac{1}{Cx^{\propto}} + \mu} dx = \int_1^n \frac{1}{Cx^{\propto}} \frac{1}{1 + \frac{\mu Cx^{\propto}}{\lambda N}} dx, \quad (16)$$

где

$$\mathrm{C} = \int_1^n \frac{1}{x^{\propto}} dx. \quad (17)$$

По результатам экспериментов в [6] было отмечено, что существует зависимость между популярностью документа и временем его обновления. В [7] было отмечено, что чем популярнее данные, тем чаще они обновляются. В [8] было предположено, что параметр модификации μ(i) зависит от популярности i-го документа.

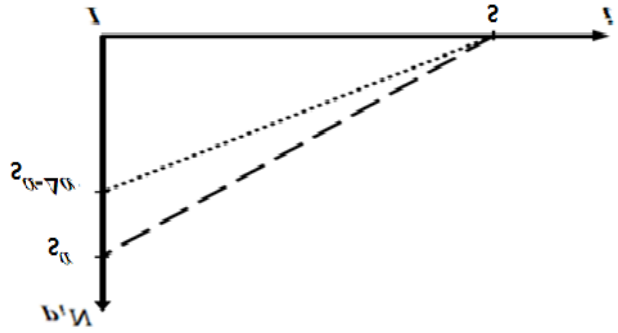

Рисунок 2. Эффект модификации

Тогда эффективность системы кэширования будет соответствовать обобщенному закону Зипфа с показателем степени α-Δα. Для определения значения Δα необходимо произвести серию экспериментов за период времени T, в ходе которых определяется ΔN – количество запросов к серверу за данными, которые успели устареть, находясь в кэше. Величина Δα вычисляется по формуле [8]:

$$\Delta\alpha = \frac{\Delta N(1-\propto)^2}{S}. \quad (18)$$

Зная значение Δα можно вычислить величину параметра модификации μ(i), как разность частот запросов для верхней и нижней кривой с рисунка 7:

$$\mu(i) = \frac{\left(\frac{S}{i}\right)^{\propto} - \left(\frac{S}{i}\right)^{\propto-\Delta\propto}}{T}. \quad (19)$$

Предсказуемо для динамической системы максимальная эффективность ниже, чем для статической системы, и может быть вычислена как:

$$\mathrm{H}_{max} \leq 2^{(\propto-1)/\propto} \frac{1-\Delta\alpha}{1-\propto}. \quad (20)$$

Размеры данных, запрашиваемых клиентами, влияют на требования к объему необходимой памяти для системы кэширования и на загрузку сети. Для эффективного функционирования системы кэширования, а также для правильного тестирования системы необходимо знать распределение размеров данных, запрашиваемых пользователями. Данные с высокой изменчивостью размеров обычно описываются распределением Парето [1]:

$$\mathrm{p}(x) = \begin{cases} \frac{ak^a}{x^{a+1}}, & x \geq k, \\ 0, & x < k \end{cases}$$

где a – параметр, определяющий форму распределения, k – параметр, определяющий масштаб распределения.

Математическое ожидание распределения Парето:

$$E(x) = \frac{ka}{a-1}, \qquad \text{для } a > 1.$$

Распределение Парето является распределением с медленно убывающим или тяжелым «хвостом», при *a* от 0 до 2. «Хвост» показывает, как медленно распределение убывает при больших значениях x. Например, рассмотрим другой вид распределения, который часто применяется в математическом анализе, а именно экспоненциальное распределение:

$$p(x) = \lambda e^{-\lambda x}.$$

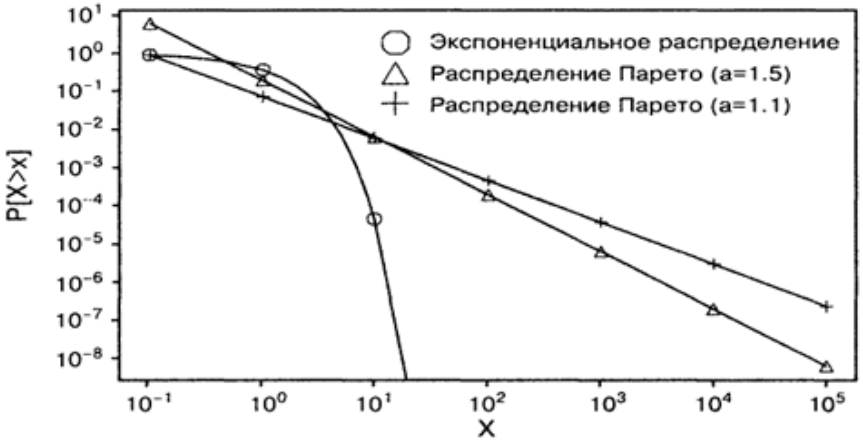

Рисунок 3. Экспоненциальное распределение и распределение Парето в логарифмическом масштабе.

На рисунке 3 показан график в логарифмическом масштабе по обеим осям распределения Парето с E(x)=1 и параметрами *a=1,1 и a=1,5*, а также график экспоненциального распределения с E(x)=1. Откуда хорошо видно, что большие значения x встречаются значительно чаще, чем в случае экспоненциального распределения. Распределение Парето в таком масштабе ведет себя линейно, наклон зависит от параметра *a.* Хотя значения *a* и отличаются от одной серии измерений к другой, но большинство исследований показало, что значение *a* находится в диапазоне от 1.0 до 1.5 [1].

Описание алгоритма

Постоянное развитие сетевых технологии обуславливает постоянное увеличение объемов информации передаваемой как через локальные сети предприятий, так и через глобальную сеть Интернет в целом. Это обстоятельство накладывает дополнительные требования на производительность сетей. Для приспособления сетей к увеличивающимся нагрузкам необходима их периодическая модернизация.

Так как увеличение производительности сети предприятия требует значительных затрат, то возникает потребность в средствах для достижения необходимой эффективности сети без модернизации сетевой инфраструктуры. Одним из возможных решений является применение технологии кэширования для снижения нагрузки на сеть, что позволит увеличить срок ее службы. Так же уменьшится время ожидания, что положительно скажется на лояльности пользователей.

В данной работе предлагается использовать разработанный алгоритм для решения данных задач. Особенностью данного алгоритма является учет нагрузки на сеть необходимой для повторной передачи данных при их вытеснении из кэша. Данные, которые значительно увеличивают нагрузку и неоднократно запрашивались, будут дольше находиться в кэше. В основе данного алгоритма лежат алгоритмы Clock, FIFO и CAR.

Поддерживаются два кольцевых списка T1, T2, с информацией о данных, которые находятся в кэше. Дополнительно поддерживаются два списка B1, B2 с информацией о документах, которые недавно были вытеснены из кэша. Списки T1 и T2 работают по алгоритму Clock, списки B1, B2 – по алгоритму FIFO.

Разделим множество всех запросов пользователей на два подмножества: «краткосрочной полезности» и «долгосрочной полезности». Данные, которые запрашивались только один раз, принадлежат первому множеству. Данные, которые запрашивались более одного раза, принадлежат второму множеству. В списке T1 могут содержаться как данные «краткосрочной полезности», так и данные «долгосрочной полезности». В списке B1 содержаться только данные

«краткосрочной полезности». В списках T2, B2 содержатся только данные «долгосрочной полезности».

Каждой записи в кэше ставится в соответствие флаг – R бит. При попадании данных в кэш, значение R бита устанавливается равным 0, запись помещается в конец списка T1. Если запрашиваемые данные уже находится в кэше, то значение R бита устанавливается равным 1.
Кроме того для каждой записи в списках T1, T2, B1, B2 поддерживается бит SB (“size bit”), который содержит информацию о сложности повторной пересылки данных. В данной работе рассматривались три варианта алгоритма, которые различались политикой изменения значения SB:

- В алгоритме test1 биту SB присваивалось значение 1, если размер соответствующих ему данных больше среднего размера запрошенных данных, иначе биту SB присваивалось значение 0.
- В алгоритме test2 биту SB присваивалось значение 1, если размер соответствующих ему данных больше медианы распределения размеров запрошенных данных, иначе биту SB присваивалось значение 0.
- В алгоритме test3 биту SB присваивалось значение 1, если размер соответствующих ему данных принадлежит интервалу (k,2k), где k – средний размер запрошенных данных, иначе биту SB присваивалось значение 0.

В начале работы алгоритма списки T1, T2, B1, B2 пустые, p=0.

От значения бита SB зависит в какую позицию списка T2 будут помещены данные. Если значение бита SB равно 1, то данные помещаются в конец списка T2. Если SB равно 0, то данные помещаются перед первой записью с SB=1, для этого поддерживается специальный указатель. Таким образом, данные, которые значительно увеличивают нагрузку и неоднократно запрашивались, будут дольше находиться в кэше.

Алгоритм является адаптивным, для этого поддерживается параметр p, который изменяется в зависимости от рабочей нагрузки. От значения параметра p зависит из какого подсписка T1 или T2 выбирается данные для удаления. Правило изменения параметра p можно сформулировать следующим образом: увеличить p, если запрашиваемая страница находиться в подсписке B1, аналогично уменьшить p, запрашиваемая страница находиться в подсписке B2.

Данные могут попасть список «долгосрочной полезности» T2 либо из списков B1, B2, то есть повторно запрашиваются данные, которые уже были в кэше, но были удалены из кэша, либо из списка T1, когда во время поиска в списке T1 кандидатов на удаление из кэша попадаются записи с битом R равным 1 они переносятся в T2 с R=0.

Описание практической части

Существуют различные показатели эффективности алгоритма кэширования, такие как время выполнения, количество запросов, для которых данные в кэше отсутствовали, объем использованного трафика и другие. В данной работе в качестве показателя эффективности алгоритма был выбран объем данных запрошенных алгоритмом. Чем меньше данный показатель, тем меньше будет нагрузка на сеть.

Для тестирования алгоритмов была разработана программа на языке C#. Последовательность запросов для каждого тестового набора задавалась случайным образом, при повторном выполнении тестового набора порядок выполнения запросов сохранялся.

Для каждого тестового набора задавались следующие параметры: общее количество запросов, количество различных запросов, размер кэша, параметр α распределения Зипфа. Тестовый запрос включал в себя номер запрашиваемой страницы и размер запрашиваемых данных.

Распределение относительной вероятности запросов в тестовых наборах соответсвует обощенному закону Зипфа. Для генерации размеров запрашиваемых данных использовалось распределение Парето с параметрами k=1 и a=1,5. Эти параметры соответсвует построенной в главе 4.1 модели. Объем данных измерялся условными единицами.

Параметры тестовых наборов приведены в таблице 1. Для сравнения с описанными алгоритмами были выбраны лежащие в их основе алгоритмы FIFO и Clock, а также алгоритмы

LRU и CAR, которые широко распространены и производительны. Размер кэша был ограничен общим объемом данных, которые можно в него поместить. Для каждого тестового набора были построены графики зависимости изменения эффективности алгоритма от размера кэша.

Таблица 1: Параметры тестовых наборов.

Название	Общее количество запросов	Количество различных запросов	Параметр α
XS1	10000	1000	0,8
XS2	10000	1000	0,75
XS3	10000	1000	0,7
S1	50000	1000	0,8
S2	50000	1000	0,75
S3	50000	1000	0,7
M1	100000	10000	0,8
M2	100000	10000	0,75
L1	500000	10000	0,8
L2	500000	10000	0,75
XL1	1000000	10000	0,8
XL2	1000000	10000	0,75

В проведенных экспериментах алгоритмом test3 были показаны наиболее стабильные результаты. При небольшой нагрузке алгоритмы test1 и test2 показали себя неплохо. Однако с ростом нагрузки их производительность ухудшается. Это связано с тем, что данные большого размера накапливаются в кэше, а данные меньшего размера быстро покидают кэш. В результате в кэше находится гораздо меньше страниц, вызывая тем самым резки рост количества обращений за данными к серверу, как следствие увеличивается нагрузка на сеть. Алгоритм test3 лишен данного недостатка.

Заключение

В данной работе были проанализированы и реализованы существующие распространенные алгоритмы кэширования. Были разработаны и реализованы алгоритмы, особенностью которых является учет нагрузки на сеть необходимой для повторной передачи данных при их вытеснении из кэша.

В ходе исследования была построена модель системы кэширования. Также было произведено тестирования алгоритмов на различных тестовых наборах. Для этого было разработано программное средство для генерации тестовых наборов. Тестовые наборы соответствовали построенной модели. Результаты тестирования показали, что алгоритмы test1 и test2 на небольших нагрузках показывают неплохой результат.

Наилучшие результаты в ходе исследования были показаны оригинальным алгоритмом test3, который превзошел все остальные алгоритмы.

Список литературы

1. Б. Кришнамурти, Дж. Рексфорд Web-протоколы. Теория и практика.// М.: ЗАО «Издательство БИНОМ», 2002 г.- с 39-129, 325-417

2. Zipf G.K. Relativity frequency as a determinant of phonetic change //Reprinted from the Harvard Studies in Classical Philology, Vol. XL, 1929

3. Steven Glaasman. A caching relay for the world wide web. // First International Conference on the WorldWide Web, CERN, Geneva, Switzerland, May 1994.

4. L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker Web Caching and Zipf-like Distribution: Evidence and Implications // IEEE Infocom, 1999 - vol. XX, no. V p.1-9

5. Долгих Д.Г., Сухов А.М. Системы резервирования трафика. Аналитическая модель // Телекоммуникации, 2007. № 3 - с. 8-11

6. Wolman A., Voelker G., Sharma N., Cardwell N., Karlin A., Levy H. On the scale and performance of cooperative Web proxy caching // Operating Systems Review 1999 -34(5) - p.16-31
7. Douglis E., Feldmann A., Krishnamurthy B., and Mogul J. Rate of change and other metric: a live study of the World Wide Web // In Proc. of the 1st USENIX Symp. on Internet Technologies and System – 1997. P. 147-158
8. Долгих Д.Г., Сухов А.М. Системы резервирования трафика. Эффект изменения документов в глобальной сети // Телекоммуникации, 2007. № 5 - с. 29-31

УДК 004.056

ФОРМАЛЬНЫЕ МЕТОДЫ ВЕРИФИКАЦИИ ПРОГРАММНЫХ СИСТЕМ НА ПРИМЕРЕ ЯЗЫКА FRAMA-C

Мұқанова Айкүн Сабитқызы
aikunmukanova@gmail.com
Магистрант 2 курса факультета «Информационные технологии» ЕНУ им.Л.Н.Гумилева, Астана, Казахстан
Научный руководитель – Сауханова Ж.С.

Введение. Программное обеспечение играет важную роль в современном мире, корректность и безопасность программного кода являются ключевыми аспектами, которые нужно учитывать при разработке программных систем. В связи с этим все большую популярность получают формальные методы верификации, которые позволяют доказывать корректность программного кода математически. Одним из инструментов для формальной верификации является язык программирования Frama-C, который предназначен для анализа и верификации программных систем на языке C. Frama-C позволяет проводить статический анализ программного кода и доказывать его корректность с помощью формальных методов. Целью данной статьи является рассмотрение принципов работы языка Frama-C и его применения для анализа и верификации программного кода на языке C. Будут рассмотрены основные концепции Frama-C, а также пример его применения для анализа корректности и безопасности программного кода.

Ключевые слова: формальные методы, верификация, программные системы, язык Frama-C, корректность ПО, безопасность.

Формальные методы верификации программных систем являются одним из важных направлений исследований в области информационных технологий. Они позволяют доказать корректность программного кода на основе формальных методов, что является необходимым условием для создания надежных и безопасных программных систем. Один из инструментов для формальной верификации программного кода - язык программирования Frama-C. Этот язык основан на языке C и используется для формальной верификации программных систем, написанных на языке C [1].

Frama-C содержит несколько модулей, каждый из которых предназначен для проверки определенного аспекта кода. Некоторые из этих модулей включают в себя:

- Value - модуль, предназначенный для анализа числовых значений переменных в программном коде. Он используется для поиска ошибок в коде, связанных с неправильным использованием переменных.
- WP (Weakest Precondition) - модуль, который позволяет автоматически генерировать условия для проверки корректности работы программного кода. Он используется для проверки корректности работы функций и алгоритмов.
- Jessie - модуль, который предназначен для проверки правильности использования указателей в программном коде. Он позволяет автоматически генерировать условия, которые проверяют правильность работы с указателями [2].

Для использования языка Frama-C необходимо создать специальный проект и загрузить в него исходный код программной системы, которую необходимо проверить. После этого можно использовать различные модули Frama-C для проверки корректности кода [3]. Процесс