Применение методов машинного обучения для выявления задач с аномальной эффективностью

Д.И. Шайхисламов

Московский государственный университет имени М.В. Ломоносова

Многие современные суперкомпьютерные приложения используют вычислительные ресурсы очень неэффективно. Для уменьшения числа таких приложений, необходимо разработать инструмент, который будет анализировать весь поток задач, исполняющихся на суперкомпьютере, и выделять среди них неэффективные запуски приложений. В данной работе рассматриваются различные методы машинного обучения с учителем для решения этой задачи. Классификация выполняется на основе различных данных системного мониторинга, таких как загрузка процессора, интенсивность работы с памятью и передачи данных по сети и т.д. По результатам проведенного сравнения на реальных данных наилучшие результаты показал алгоритм «случайный лес». Представленная в этой работе модель классификатора в настоящее время функционирует на суперкомпьютере «Ломоносов».

Ключевые слова: суперкомпьютер, высокопроизводительные вычисления, поток задач, машинное обучение, эффективность, анализ приложений

1. Введение

Администраторам суперкомпьютера очень важно то, чтобы ресурсы суперкомпьютера использовались эффективно. Неэффективное использование ведет к простою ресурсов, в то время как эти ресурсы мог использовать какой-нибудь другой пользователь. Также это приводит к тому, что пользователь будет позже получать результаты работы приложения из-за замедления его работы. Таким образом, возрастает актуальность задачи выявления неэффективных приложений. Для решения данной проблемы авторами статьи ведется разработка инструмента, который, анализируя поток исполняющихся приложений суперкомпьютера, выявляет среди них неэффективные запуски приложений и оповещает пользователей об этих запусках.

На данный момент существуют решения, которые позволяют анализировать поведение отдельной программы (такие как Scalasca, Vampir, HPCToolkit), но они требуют непосредственного вмешательства пользователя, а из-за того, что пользователи зачастую не подозревают о том, что их приложение может работать неэффективно, такие приложения редко используются. Также существуют инструменты, такие как NuPIC[1] или Rocana[2], которые выявляют аномалии в потоке данных в реальном времени, но они анализируют поток данных, представляемых как {время : значение}. Это можно использовать при анализе одной динамической характеристики (таких как загруженность процессора, интенсивность работы с памятью и т.д.), и эти приложения будут выявлять аномалии, но они будут показывать только факт того, что произошло что-то необычное, а в будущем нам интересно еще выяснять первопричину этой аномалии. Также данные инструменты могут анализировать поток только одной динамической характеристики, но некоторые аномалии выявляются только по совокупности динамических характеристик. По этим причинам было решено, что данные инструменты не подходят для решения нашей задачи.

Был выполнен поиск работ, направленных на анализ потока суперкомпьютерных приложений в реальном времени с целью определения неэффективных запусков, однако исследований именно по данной теме не было найдено. Но есть работы по применению методов машинного обучения для анализа приложений, например [3,4]. К примеру, в работе[5] использовались методы машинного обучения с учителем для классификации того, какое именно приложение было запущено. В рамках данной работы для выбора подходящего алгоритма авторы проверяли алгоритмы на задаче нахождения неэффективных приложений в некотором большом наборе этих приложений. Но авторы использовали свой критерий того, что

приложение было аномальным. Был применен выбор неэффективных приложений по порогам, то есть приложение считалось неэффективным, если значение динамической характеристики было больше или меньше некоторого заранее заданного числа. Например, они использовали следующие пороги: загрузка процессора < 30, число тактов на инструкцию < 2 и т.д. Пороговое выделение аномалий, безусловно, может использоваться, но это будет выявлять только тривиальные случаи, в то время как нам хотелось бы выделять как можно больше аномальных случаев.

2. Определение понятия аномалии

Первым шагом было определение понятия аномалии. Аномалией будем называть приложение, которое так эффективно или так неэффективно использует ресурсы суперкомпьютера, что оно начинает выделяться среди других приложений. Однако найти точный критерий аномальности приложения очень тяжело. Это связано, в основном, с тем, что на суперкомпьютере выполняется очень большое количество приложений, различных как по свойствам, так и по типу решаемых задач. Именно это стало основной причиной использования машинного обучения.

Следующим шагом было выделение типов аномалий. Это было необходимо для понимания того, что именно будет оцениваться. Было выделено несколько независимых классификаций для аномалий.

Первая классификация: «аномально эффективная» и «аномально неэффективная». Аномально эффективной задачей будем называть приложение, которое гораздо эффективнее остальных приложений использует ресурсы суперкомпьютера. Например, приложение было спроектировано таким образом, чтобы минимизировать любые накладные расходы, оптимально распределить ресурсы по ядрам и т.д. Было бы интересно выделять не только неэффективные приложения, но и эффективные, для их последующего анализа и выяснения причин, почему приложение так хорошо приспособлено к данной суперкомпьютерной системе. После выяснения причин, можно составить список рекомендаций для пользователей, как писать более эффективные приложения.

Аномально неэффективной будем называть приложение, которые настолько неэффективно используют ресурсы суперкомпьютера, что это может быть нами замечено (например, в общем потоке приложений или в потоке приложений пользователя). Это может возникнуть по разным причинам: зависание или зацикливание программы, непродуманный алгоритм, неэффективная программная реализация и т.д.

Вторая классификация аномальных приложений: «в общем потоке приложений», «в потоке приложений пользователя» или «в рамках одного приложения». Аномалией в общем потоке приложений будем называть аномально эффективную или аномально неэффективную программу, которая была замечена при анализе всего потока приложений суперкомпьютера. Аномалия в потоке приложений пользователя отличается от аномалии в общем потоке приложений только тем, что эта аномалия выявляется при анализе приложений только конкретного пользователя. Выделять этот тип аномалий нам было бы интересно, потому что очень часто поведение приложений пользователя примерно одинаково. Если одно из приложений не похоже на остальные, зачастую это значит, что оно запущено некорректно или в процессе его выполнения возникли какие-то ошибки. Еще одной причиной, по которой этот тип аномалий выделяется от остальных, является то, что те приложения, которые считаются аномальными в потоке приложения пользователя, могут быть совершенно обычными в общем потоке. Отклонение приложения от среднестатистических приложений пользователя может быть вызвано многими возможными причинами (например, ошибкой в программе или некорректными входными данными), что делает его уже потенциально аномальным. На рисунке 1 приведен пример аномалий в потоке приложений пользователя.

Рис. 1. Пример аномалий в потоке приложений пользователя

Когда речь идет об аномалии в рамках одного приложения, то рассматривается приложение отдельно от всего потока, не учитываются характеристики других приложений, а рассматривается только поведение программы во время выполнения. Анализ поведения программы во время выполнения позволит получать дополнительные зависимости между динамическими характеристиками при разных типах неэффективных задач, что поможет определить причину появления данной аномалии.

В данной работе было решено выделять аномально неэффективные приложения в общем потоке задач.

3. Применение машинного обучения

Для применения машинного обучения, необходимо представлять приложения в унифицированном формате. Все данные о приложениях, запускаемых на суперкомпьютере, собираются системой мониторинга суперкомпьютера. В данном случае, использовались данные системы мониторинга суперкомпьютера «Ломоносов». Были доступны следующие динамические характеристики, которые в дальнейшем использовались:

- Промахи при обращении в кэш-память первого уровня (cache_1)
- Промахи при обращении в кэш-память третьего уровня (cache_3)
- Средняя загрузка системы (Loadavg)
- Загрузка процессора (cpu_user)
- Количество обращений в память (mem_load)
- Количество записей в память (mem_store)
- Количество полученных байтов в секунду по Infiniband (ib_rcv_data)
- Количество полученных пакетов в секунду по Infiniband (ib_rcv_pckts)
- Количество отправленных байтов в секунду по Infiniband (ib_xmit_data)
- Количество отправленных пакетов в секунду по Infiniband (ib_xmit_pckts)

Так как каждая динамическая характеристика по каждой задаче представляет временной ряд зачастую большого объема, для упрощения задачи было решено рассматривать интегральные характеристики по всей задаче вместо временных рядов. Также было принято решение каждую динамическую характеристику представлять двумя значениями – медианой и коэффициентом осцилляции. Рассматривались такие кандидаты, как среднее значение, квантили, дисперсия. Медиана была выбрана, т.к. эта характеристика показывает, не выше или не ниже какого уровня приложение работало половину времени выполнения. Эта характеристика временного ряда нечувствительна к выбросам во время выполнения, что очень важно в нашем случае, т.к. иногда датчики, по которым система мониторинга собирает данные, показывают результаты, которые не соответствуют действительности, и это может испортить выборку. Коэффициент осцилляции – относительная характеристика, которая показывает, как

сильно крайние значения колеблются вокруг среднего значения. Зачастую, чем коэффициент выше, тем это хуже, т.к. это будет показывать, что либо какой-то узел работает менее эффективно, чем среднестатистический узел, либо какой-то узел нагружен сильнее, чем другие. В результате использование именно медианы и коэффициента осцилляции дало наибольшую точность классификации.

Для решения нашей задачи было решено сравнить следующие алгоритмы машинного обучения с учителем: линейный дискриминантный анализ[6], дерево принятия решений (Decision Tree) и случайный лес[8] (Random Forest). Дерево принятия решений – это группа алгоритмов классификации, в данной работе используется алгоритм CART[7]. Были испробованы также и другие методы классификации, например, наивный байесовский классификатор или AdaBoost, но они показали плохие результаты.

Обучение проводилось на 300 вручную классифицированных приложениях, и были выделены 3 группы приложений: обычные (normal), аномальные (abnormal) и подозрительные (suspicious). В группу обычных вошли 110 приложений, в группу аномальных – 130 приложений, в группу подозрительных – 60 приложений. В группу «подозрительные» были внесены те приложения, для которых тяжело однозначно определить, обычное это приложение или аномальное.

В дальнейшем, точностью будем называть величину, вычисляемую по следующей формуле:

$$\text{Точность} = \frac{\text{Количество правильно классифицированных элементов}}{\text{Общее количество элементов}}.$$

Для оценки точности выбранных алгоритмов проводилась проверка точности на тестовой выборке. Обучающая выборка случайным образом делилась на 4 части, одна из которых считалась тестовой выборкой. Оставшиеся элементы в обучающей выборке еще раз делились на 4 части. На трех частях модель обучалась, а на четвертой вычислялась точность. Проверка проходила для каждой из 4 частей. В итоге получались 4 точности, для каждой части, выбранной для проверки. Далее выбиралась модель, точность у которой была наибольшей. После этого вычислялась точность на тестовой выборке. В последующем точностью классификации будем считать точность на тестовой выборке.

В итоге получили следующие результаты:

- Линейный дискриминантный анализ: 0,74 на обучающей выборке.
- Дерево принятия решений: 0,75 на тестовой выборке.
- Случайный лес: 0,82 на тестовой выборке.

Линейный дискриминантный анализ даже на обучающей выборке показал низкие результаты, из-за чего этот метод был отброшен. Дерево принятия решений и случайный лес показали относительно неплохие результаты, но случайный лес всегда опережал дерево принятия решений, из-за чего был выбран именно этот алгоритм. Еще одной особенностью случайного леса является то, что алгоритм является группой деревьев принятия решений, а классификация производится «голосованием» - каждое дерево классифицирует поданное приложение. Исходя из того, какой класс был выбран наибольшее число раз, и будет классифицировано приложение. Но, в отличие от дерева принятия решений, которое только выдает класс, которым было помечено приложение, случайный лес еще выдает, за какой класс сколько деревьев проголосовало. Используя эти данные, модель строит «вероятности» того, что элемент относится к тому или иному классу, которые высчитываются следующим образом:

$$p_i = \frac{\text{количество деревьев, которые отнесли элемент к классу } i}{\text{общее количество деревьев}}.$$

Так как случайный лес является группой деревьев принятия решений, то немаловажным параметром алгоритма является количество деревьев. Мы провели серию экспериментов для выяснения оптимального числа деревьев. Было выявлено, что точность увеличивается при увеличении числа деревьев до 256, а после точность остается на одном уровне. Поэтому параметр числа деревьев был установлен как 256.

На рис. 2 и 3 представлены графики ROC-кривых[9] для дерева принятия решений и случайного леса. Эти графики показывают, как хорошо классификатор классифицирует тот или иной класс задач. Чем больше площадь под кривой – тем лучше.

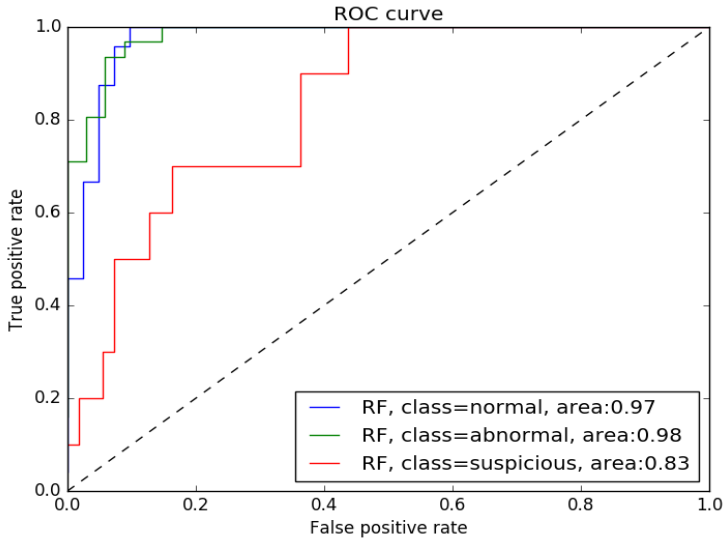


Рис. 2. ROC-кривая для модели классификатора «случайный лес»

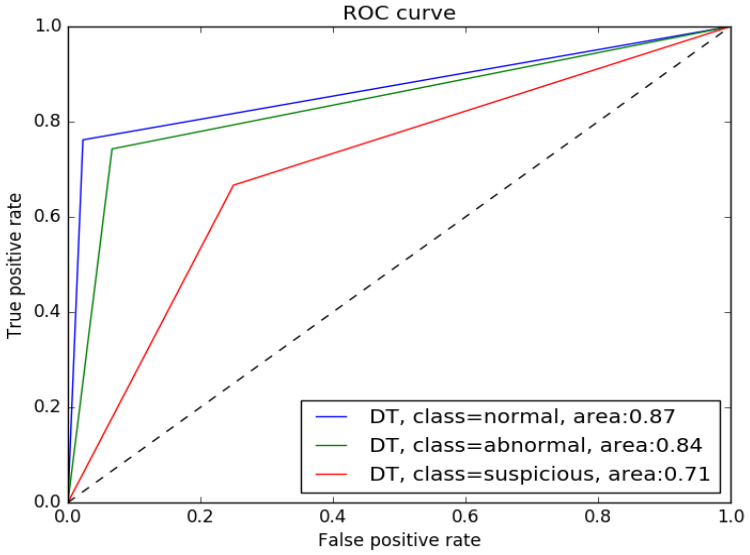


Рис. 3. ROC-кривая для модели классификатора «дерево принятия решений»

Можно заметить тенденцию для обоих классификаторов: и случайный лес, и дерево принятия решений классифицируют класс подозрительных задач хуже, чем класс аномальных и обычных задач. Проанализировав распределения динамических характеристик приложений разных групп, был сделан вывод, что это связано с тем, что в классе подозрительных приложений много элементов, которые близки к аномальным или обычным приложениям, из-за чего классификатор их относит в ту или иную группу. Это приводит к ошибкам классификации и снижению точности. То, что класс подозрительных задач классифицируется хуже других классов, хорошо видно по матрице ошибок (confusion matrix), приведенной в таблице 1. Матрица ошибок была построена для тестовой выборки.

Таблица 1. Матрица ошибок для случайного леса

		Классифицированы как		
		Обычные	**Аномальные**	**Подозрительные**
Реальные классы	**Обычные**	25	0	2
	Аномальные	1	35	1
	Подозрительные	3	4	6

Обучающая выборка строилась итерационно, т.е. задачи постепенно включались в выборку. Во время этого процесса было обнаружено, что после того, как количество элементов в обучающей выборке превысило число 100, точность классификации оставалась примерно на одном уровне. Это хорошо видно на рисунке 4, в котором показан график кривой обучения. Он показывает зависимость точности модели классификатора от количества элементов в обучающей выборке.

Рис. 4. График кривой обучения для модели классификатора «случайный лес»

Видно, что после 100 элементов точность остается примерно на одном и том же уровне. Возникают следующие вопросы: из-за чего это может быть и как можно увеличить точность. Ответ на первый вопрос дать тяжело, но самым вероятным ответом будет то, что это связано со сложностью классификации подозрительных задач. В этом не поможет увеличение числа элементов в выборке, т.к. даже тогда приложения из класса подозрительных задач будут похожи на приложения из класса обычных или аномальных.

Следующий шаг – увеличение точности модели. Были испробованы следующие методы:

- Уменьшение размерности элементов, т.е. использование только части доступных динамических характеристик.
- Использование производных характеристик

Было использовано уменьшение размерности элементов, потому что это может помочь при чрезмерной аппроксимации модели. Чрезмерная аппроксимация модели означает, что модель так сильно подстраивается под обучающую выборку, что элементы из этой выборки она классифицирует хорошо, а реальные примеры классифицирует намного хуже. Для уменьшения размерности был применен линейный дискриминантный анализ. Этот метод показал, что все характеристики важны для классификации, из-за чего нет возможности уменьшить размерность элементов без уменьшения точности. Например, уменьшение размерности этим методом в два раза привело к уменьшению точности классификатора с 0,82 до 0,76.

Далее было решено попробовать выделить производные характеристики, которые позволяют лучше различать приложения разных классов. Производной характеристикой будем называть такую характеристику, которая была получена путем алгебраических преобразований имеющихся динамических характеристик. Возникает вопрос, какие производные характеристики стоит использовать? Есть 2 способа: вручную подобрать производные характеристики, которые могут увеличить точность, или же перебрать производные

характеристики определенного вида и выбрать те, которые дают наибольшую точность. Были испробованы оба варианта. Были добавлены 5 вручную подобранных производных характеристик, и данный способ позволил нам увеличить точность классификатора до 0,835. Были добавлены следующие характеристики:

- ln(IB_rcv_pckts_median/IB_rcv_pckts_oscil)
- ib_rcv_pckts_median/ib_xmit_pckts_median
- cpu_user_median/cpu_user_oscil
- ib_xmit_data_median/ib_xmit_pckts_median
- ln(cache_3_median/cache_3_oscil)

Также был сделан перебор всех производных характеристик вида {характеристика №1}/{характеристика №2}. Если выбрать одну наилучшую производную характеристику, то была получена точность 0,83, а если использовать 2 наилучшие производные характеристики, то точность повышалась до 0,845. Перебрать больше двух производных характеристик не представлялось возможным, т.к. сложность перебора возрастает экспоненциально. Результаты получились хорошими, но было принято решение остаться на тех производных характеристиках, которые были подобраны вручную, т.к. есть понимание того, какие реальные свойства приложений они отражают. В случае же производных характеристик, полученных способом перебора, тяжело объяснить, почему именно данная характеристика так увеличивает точность. К примеру, пока не удалось объяснить, почему характеристика ib_xmit_data_median/loadavg_oscil увеличивает точность классификации.

В итоге, на данный момент используется модель классификатора случайный лес, использующий вручную подобранные производные характеристики, с точностью на тестовой выборке в 0,835.

4. Реализация и апробация предложенного метода

Схема разработанного программного комплекса показана на рис. 5.

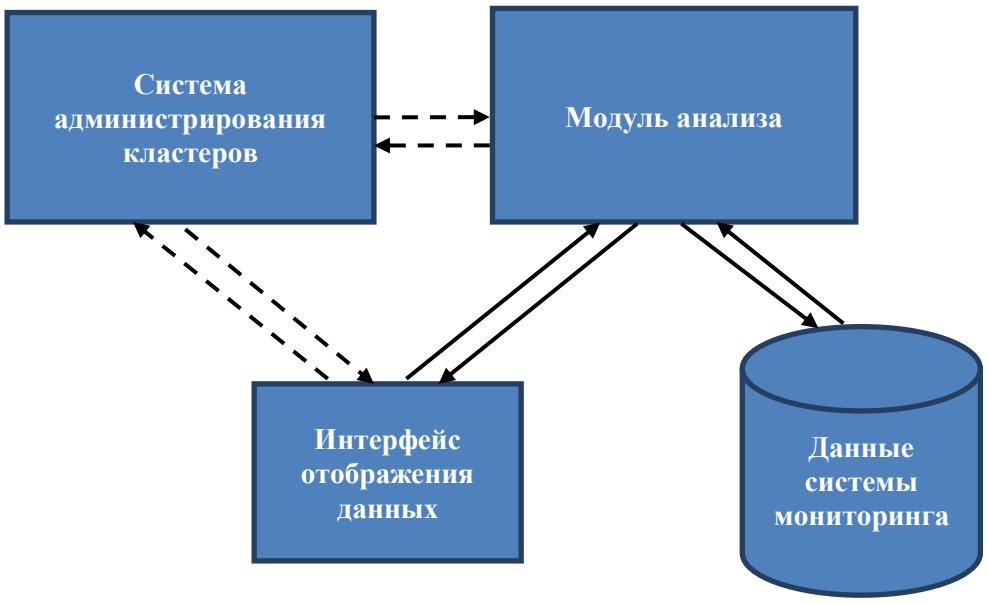


Рис. 5. Схема программного комплекса. Сплошные линии - реализованные связки между компонентами, пунктирные линии - не реализованные связки.

Были разработаны модуль анализа и интерфейс отображения данных. В данном случае, системой администрирования кластеров является Octoshell[10]. Связка с системой администрирования кластеров нам понадобится на последующих стадиях работы, когда нужно будет оповещать пользователей об аномальных задачах.

Программная часть, описанная в данной работе, была реализована с использованием языка программирования Python 2.7. Используется библиотека scikit-learn[11], где реализованы алгоритмы машинного обучения с учителем. Рассмотрим подробнее алгоритм, как приложение проходит весь цикл от конца сбора данных системой мониторинга до определения класса, к которому относится данная задача:

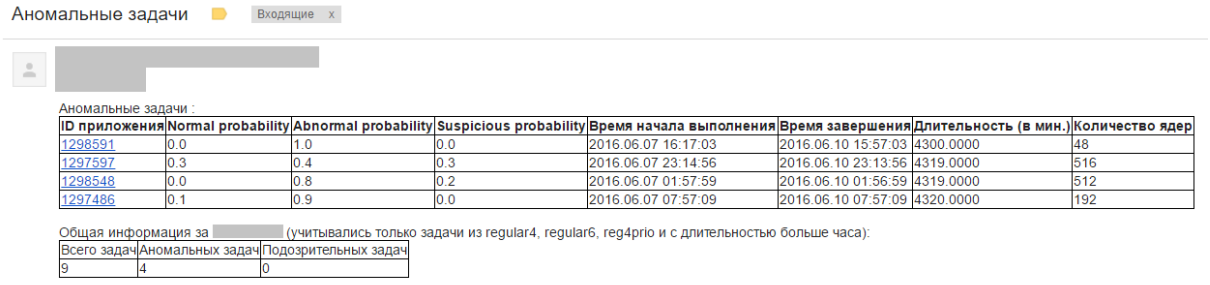
a) На суперкомпьютере «Ломоносов» функционирует система мониторинга, которая собирает данные по всем приложениям, выполняющимся в данный момент. После завершения работы приложения, в специальную таблицу базы данных системы мониторинга заносится запись с базовой информацией об этом приложении. Например, ID приложения, время начала выполнения. Все данные системы мониторинга хранятся в «сыром», то есть необработанном, виде. Данные по каждой динамической характеристике хранятся в виде <метка времени>;<узел>;<среднее>;<максимальное значение>;<минимальное значение>. На данный момент данные хранятся за каждые 5 минут работы суперкомпьютера, т.е. среднее, максимальное и минимальное значения для каждой характеристики по каждому узлу высчитываются за промежуток времени в 5 минут.

b) Каждые полчаса наша программа сканирует таблицу БД для выявления новых завершившихся приложений. После этого из сырых данных отбираются нужные данные по каждому приложению, по каждой динамической характеристике высчитывается медиана, среднее, максимальное и минимальное значения. Полученные данные сохраняются в БД модуля анализа данных.

c) Каждая новая полученная задача проходит через обученную модель классификатора, и модель дает каждой задаче класс, к которому она относится.

Разработанная система на данный момент запущена на суперкомпьютере «Ломоносов», классификация происходит в реальном времени. Так как у нас идет классификация каждые 30 минут, количество приложений за это время невелико (максимум 20), и сама классификация выполняется за очень короткое время (доли секунд).

В зависимости от того, к какому классу было соотнесено каждое из данных приложений, формируется список задач, который каждый день отсылается по электронной почте администраторам. В список включаются все аномальные задачи, а также самые подозрительные приложения. Пример такого отчета можно приведен ниже (см. рисунок 6). На этом рисунке мы видим столбцы Normal probability, Abnormal probability и Suspicious probability, которые показывают коэффициенты вероятности отнесения данной задачи к определенному классу.


Аномальные задачи Входящие x

Аномальные задачи :

ID приложения	Normal probability	Abnormal probability	Suspicious probability	Время начала выполнения	Время завершения	Длительность (в мин.)	Количество ядер
1298591	0.0	1.0	0.0	2016.06.07 16:17:03	2016.06.10 15:57:03	4300.0000	48
1297597	0.3	0.4	0.3	2016.06.07 23:14:56	2016.06.10 23:13:56	4319.0000	516
1298548	0.0	0.8	0.2	2016.06.07 01:57:59	2016.06.10 01:56:59	4319.0000	512
1297486	0.1	0.9	0.0	2016.06.07 07:57:09	2016.06.10 07:57:09	4320.0000	192

Общая информация за (учитывались только задачи из regular4, regular6, reg4prio и с длительностью больше часа):

Всего задач	Аномальных задач	Подозрительных задач
9	4	0

Рис. 6. Пример отчета о найденных за сутки аномалиях

Приведу пример выделенной аномалии. На рис. 7 и 8 приведены графики промахов при обращении в кэш-память третьего уровня и количества записей в память в секунду соответственно. Данные графики были получены средствами инструмента автоматической генерации отчетов по приложениям JobDigest[12], который также использует данные системы мониторинга. JobDigest очень сильно помог на стадии набора приложений для обучающей выборки.

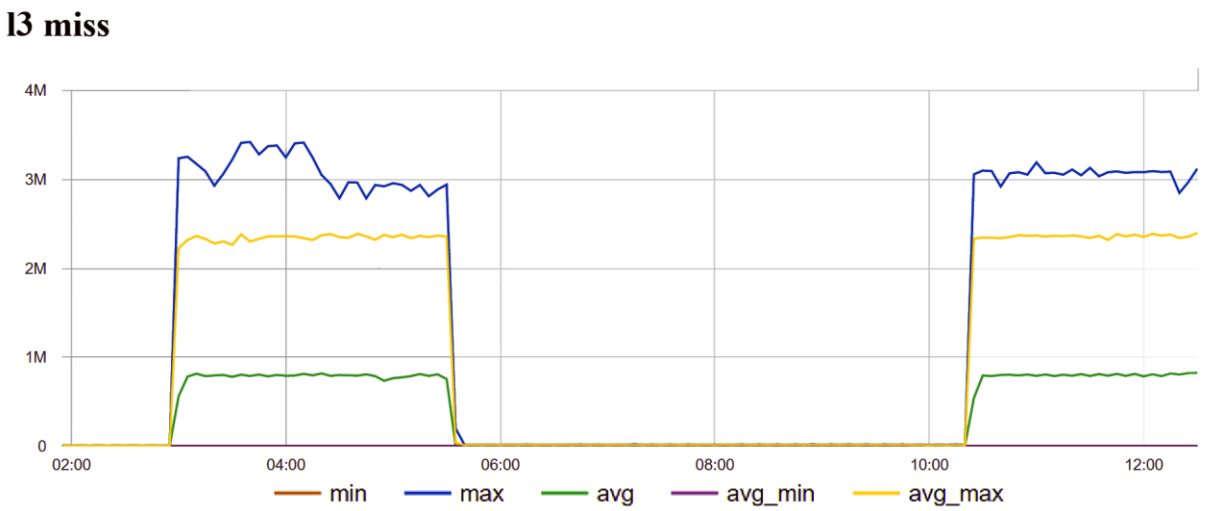


Рис. 7. График количества промахов при обращении в кэш-память третьего уровня в секунду

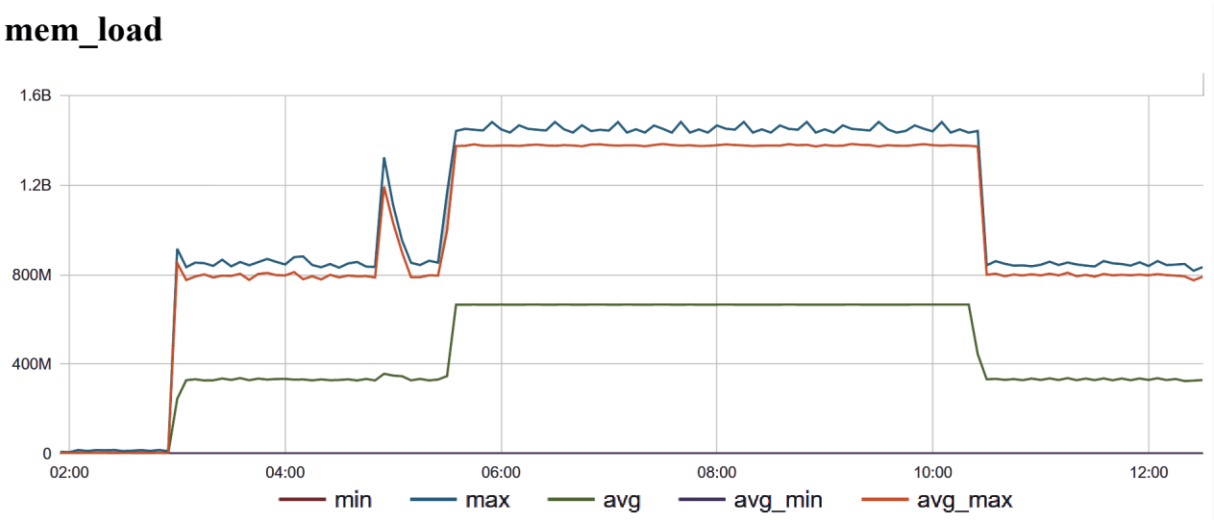


Рис. 8. График количества обращений в память в секунду

Почти все оставшиеся динамические характеристики ведут себя как количество промахов при обращении в кэш-память третьего уровня. Можно сразу заметить, что приложение ведет себя необычно. Есть большой промежуток времени (несколько часов), когда приложение интенсивно работает с памятью, но при этом нет промахов в кэш-память. Всегда есть вероятность того, что это такое приложение, у которого данные помещаются в кэш-память узлов и обращений в ОЗУ не требуется, однако гораздо больше это похоже на результат возникновения ошибок при выполнении. Такое поведение как минимум подозрительно, и надо дать пользователю знать, что его приложение может работать некорректно.

5. Заключение и планы на будущее

В рамках данной работы был разработан инструмент, который анализирует поток запускаемых приложения суперкомпьютера «Ломоносов» и выявляет неэффективные запуски приложений. Информация обо всех неэффективных приложениях отсылается администраторам по электронной почте. На текущий момент система функционирует на суперкомпьютере «Ломоносове» и показывает достаточно высокую точность, в частности, все выявленные приложения действительно являются аномальными или подозрительными.

На данный момент выявляется только факт того, что приложение аномальное. Планируется в будущем продолжать работу в этом направлении, и не только выявлять, но и находить причину появления той или иной аномалии. Также мы рассматривали только аномалии в

общем потоке запускаемых приложений, планируется также разработать инструмент для выявления аномалий в потоке пользователя.

Выражаю благодарность моему научному руководителю Воеводину Вадиму Владимировичу за помощь при написании данной статьи. Данное исследование проводится в рамках проекта, поддержанного стипендией Президента РФ (СП-1981.2016.5), а также при финансовой поддержке РФФИ, грант №16-07-00972.

Литература

1. NuPIC: Numenta Platform for Intelligent Computing. URL: http://numenta.org (дата обращения: 06.06.2016).
2. Rocana: Anomaly Detection. URL: https://www.rocana.com/products/technology/advanced-analytics-anomaly-detection (дата обращения: 10.06.2016).
3. Aleem S., Capretz L.F., Ahmed F. Benchmarking machine learning techniques for software defect detection // International Journal of Software Engineering & Applications. 2015. V. 6, №3.
4. Сиднев А. А., Гергель В. П. Автоматический выбор наиболее эффективных реализаций алгоритмов // Вычислительные методы и программирование. 2014. Т. 15, № 4. С. 579–592.
5. Steven M. Gallo, Joseph P. White Analysis of XDMoD/SUPReMM Data Using Machine Learning Techniques // 2015 IEEE International Conference on Cluster Computing. 2015. P. 642-649.
6. T. Hastie, R. Tibshirani, J. Friedman. The Elements of Statistical Learning. P. 106-119, 2008.
7. Leo Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone. Classification and regression trees. 1984.
8. Leo Breiman. Random Forests // Machine Learning. 2001. V. 45, № 1, P. 5-32.
9. Receiver operating characteristic. URL: https://en.wikipedia.org/wiki/Receiver_operating_characteristic (дата обращения: 21.11.2015).
10. Octoshell: система управления доступом к Суперкомпьютерному комплексу МГУ им. М.В. Ломоносова. URL: http://users.parallel.ru/ (дата обращения: 10.06.2016).
11. Scikit-learn, machine learning in Python. URL: http://scikit-learn.org/stable/ (дата обращения: 15.09.2015).
12. JobDigest – подход к исследованию динамических свойств задач на суперкомпьютерных системах / А. В. Адинец, П. А. Брызгалов, В. В. Воеводин и др. // Вестник Уфимского государственного авиационного технического университета. – 2013. – Т. 17, № 2 (55). – С.131–137.

Using machine learning methods to detect applications with abnormal efficiency

D.I. Shaykhislamov

Lomonosov Moscow State University

At the moment a lot of supercomputing applications are inefficient in terms of usage of available resources. To decrease the number of such inefficient applications, a tool for supercomputer task flow analysis and detection of inefficient launches is needed. In this paper several supervised machine learning methods are considered. The classification is based on system monitoring data (e.g. CPU load, network usage etc.). The experiments on “Lomonosov” supercomputer showed that Random Forest algorithm is the best option to accomplish given goal. Also several data modifications were made to increase the accuracy of classification. At the moment the resulting classifier model is working and being tested on “Lomonosov” supercomputer. The experiment results demonstrating the efficiency of the resulting model are also included in this paper.

Keywords: supercomputer, anomaly detection, program efficiency, machine learning

References

1. NuPIC : Numenta Platform for Intelligent Computing. URL: http://numenta.org (accessed: 06.06.2016).
2. Rocana : Anomaly Detection. URL: https://www.rocana.com/products/technology/advanced-analytics-anomaly-detection (accessed: 10.06.2016).
3. Aleem S., Capretz L.F., Ahmed F. Benchmarking machine learning techniques for software defect detection // International Journal of Software Engineering & Applications. 2015. V. 6, №3.
4. Sidnev A.A., Gergel V.P. Automatic selection of the fastest algorithm implementations // Vychislitel'nye Metody i Programmirovanie. 2014. V.15, № 4. P. 579–592.
5. Steven M. Gallo, Joseph P. White Analysis of XDMoD/SUPReMM Data Using Machine Learning Techniques // 2015 IEEE International Conference on Cluster Computing. 2015. P. 642-649.
6. T. Hastie, R. Tibshirani, J. Friedman. The Elements of Statistical Learning. P. 106-119, 2008.
7. Leo Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone. Classification and regression trees. 1984.
8. Leo Breiman. Random Forests // Machine Learning. 2001. V. 45, № 1, P. 5-32.
9. Receiver operating characteristic. URL : https://en.wikipedia.org/wiki/Receiver_operating_characteristic (accessed: 21.11.2015).
10. Nikitenko D. A., Voevodin V. V., Zhumatiy S. A. Octoshell: Large supercomputer complex administration system // russian supercomputing days international conference // Russian Supercomputing Days International Conference, Moscow, Russian Federation, 28-29 September, 2015, Proceedings. — Vol. 1482. — CEUR Workshop Proceedings, Moscow, 2015. — P. 69–83.
11. Scikit-learn, machine learning in Python. URL: http://scikit-learn.org/stable/ (accessed: 15.09.2015).
12. JobDigest – approach to jobs dynamic properties investigation on supercomputer systems / A. V. Adinetz, P. A. Bryzgalov, V. V. Voevodin etc. // Vestnik UGATU. – 2013. – Vol. 17, No 2 (55). – P. 131–137.