УДК 004.4

И.А. Вавилин, А.Н. Глибовец, Н.Н. Глибовец

Разработка поискового робота на *Erlang*

Описано использование преимуществ функционального языка программирования *Erlang* для построения распределенных вычислений в веб-среде на примере разработки поискового робота.

This paper describes the advantages of using a functional programming language *Erlang* for building a distributed computing in the Web environment on the example of a search engine robot.

Описано використання переваг функціональної мови програмування *Erlang* для побудови розподілених обчислень у веб-середовищі на прикладі розробки пошукового робота.

Введение. В современном мире распределенные вычисления – актуальное направление развития компьютерных наук. Существует много задач, требующих сложных или длительных вычислений [1]. Для работ такого масштаба не достаточно мощностей одного вычислительного узла, даже если это современный суперкомпьютер. Поэтому вариантом решения этой проблемы есть разбиение задачи на части и использование сети компьютеров. Для управления такими сетями создают специализированные программные комплексы. Большинство из них разработаны на *C++*, поскольку это язык высокого быстродействия, поддерживающий мощную библиотеку для организации распределенных вычислений – *Message Passing Interface* (*MPI*, интерфейс передачи сообщений). Это хорошо стандартизованный механизм построения параллельных программ, основанный на модели обмена сообщениями. Известны и системы, реализованные на других объектно-ориентированных языках *Java* и *PHP* [2]. В последнее десятилетие появились функциональные языки программирования со специализированными средствами разработки распределенных вычислительных сред [3, 4].

В веб-среде проблема масштабных вычислений также наблюдается. Количество информации в сети Интернет очень быстро растет и для ее обработки и структурирования разрабатываются специальные программные системы, называемые поисковыми роботами (*web crawler*, *spider*) [5].

Наиболее известные поисковые системы имеют своих поисковых роботов. Большинство из них разработаны на *C++*, *Java* и *PHP* [6]. Но неизвестны разработки роботов с использованием функциональных языков, хотя они имеют определенные преимущества для решения этой задачи. Поэтому цель этой работы – выявление этих преимуществ на примере создания поискового робота с использованием функционального языка *Erlang*.

Erlang – функциональный язык программирования с динамической типизацией, разработанный для создания распределенных вычислительных систем [7]. Он имеет средства для порождения параллельных процессов и их коммуникации с помощью асинхронных сообщений. Кросс-платформенность обеспечивается путем компиляции программы в байт-код, выполняемый на виртуальной машине.

Поисковый робот

Поисковый робот (ПР) – это программная компонента, входящая в состав поисковой системы, предназначенная для обработки веб-страниц и занесения информации о них в базу данных поисковой системы [8]. Стандартные функции ПР – анализ структуры сайта, сохранение в базе данных информации о каждой странице сайта и мониторинг его доступности.

Некоторые ПР имеют набор дополнительных функций, среди которых можно выделить: сохранение локальной копии всех обработанных страниц для обеспечения быстрого поиска ин-

формации в них, валидация *HTML*-кода, сбор *e-mail* адресов для отправки спама [9].

Алгоритм работы ПР начинается с анализа некоторого начального набора веб-адресов. На каждой выбранной странице ищутся новые ссылки, которые хранятся в очереди для дальнейшей обработки.

Обработка веб-адресов регулируется определенным набором политик: выбора страниц для загрузки, время повторной обработки веб-страницы для проверки изменений, которые могли произойти после последней обработки, вежливости (регулирует предотвращение перегрузки обрабатываемых сайтов), политика параллелизации (регулирует механизм координации действий для распределенных поисковых роботов).

Во избежание обработки одних и тех же страниц используется нормализация веб-адресов. Это приведение ссылки к определенному каноническому виду. Для этого, в частности, символы приводятся к одному регистру, убираются номера портов по умолчанию, лишние точки и слеши. В разработанном ПР реализована функция нормализации адресов, выполняющая все основные операции без изменения семантики адреса.

Архитектура поискового робота

Поисковый робот был разработан по принципу клиент-серверной архитектуры. Он состоит из главного процесса, который получает задания извне и распределяет их между рабочими процессами. Главный процесс периодически проверяет наличие новых задач в базе данных. Рабочие процессы могут быть запущены на любом, правильно настроенном компьютере, к которому есть доступ по сети. Рабочие процессы также могут быть запущены на том же компьютере, что и главный процесс. Тогда поисковый робот перестает быть распределенным, но сохраняет полную функциональность. Для хранения информации используется *NoSQL* СУБД *MongoDB*, которая тоже может работать в распределенном режиме. Она имеет высокую производительность, а ее документоориентированность подходит для сохранения результатов работы поискового робота (рис. 1).

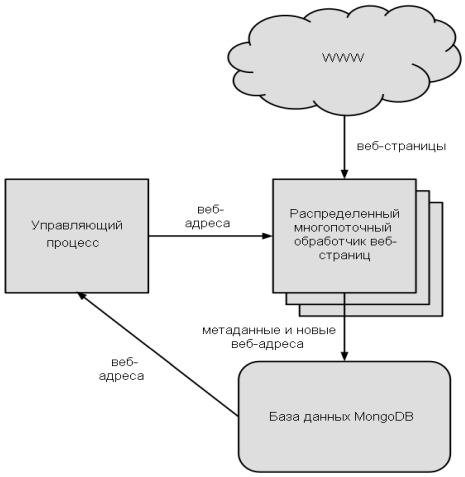

Рис. 1. Архитектура поискового работа

СУБД *MongoDB*

В качестве хранилища данных для поискового работа была выбрана СУБД *MongoDB*. Она документо-ориентированна и относится к классу *NoSQL* СУБД. Данные сохраняются в формате *BSON* (похожий на *JSON*) и не имеют определенной схемы, т.е. документы в коллекции могут иметь разную структуру.

Выделяют следующие свойства *MongoDB* [13]: гибкий язык для формирования запросов (можно написать собственные функции на *Javascript* для обработки данных); полная поддержка индексов (любое поле может быть проиндексировано); поддержка *master–slave* репликации (*slave*-узлы могут выбрать новый *master*-узел, если текущий недоступен); горизонтальная масштабируемость (с использованием шардинга); поддержка балансировки нагрузок между шардами; данные можно агрегировать, используя подход *MapReduce*.

Структура базы данных *crawler* для разработанного поискового робота состоит из трех коллекций. Для хранения веб-адресов, требующих обработки, используется коллекция *pending_urls*. Документы в этой коллекции обычно (поскольку СУБД не накладывает ограничений) имеют структуру, приведенную в листинге 1. Кроме самой ссылки в них хранится список рефералов и статус документа – в ожидании или в обработке. Также может быть заполнено поле *operation*. В этом случае будет проведена дополнительная обработка документа по типу операции.

"*_id*" : *ObjectId*,
"*url*" : *string*,
"*referral_list*" : *array*,
"*status*" : *string*,
"*d_change*" : *date*,
"*operation*" : *array*

Листинг 1. Структура документа в коллекции *pending_urls*

Для хранения результатов обработки веб-страниц используется коллекция *processed_urls*. Структура документов в этой коллекции приведена в листинге 2.

"*_id*" : *string*,
"*content-length*" : *string*,
"*content-type*" : *string*,
"*server*" : *string*,
"*http_returncode*" : *integer*,
"*referral_count*" : *integer*,
"*referral_list*" : *array*,
"*links_count*" : *integer*,
"*d_processed*" : *date*,
"*operation*" : *array*

Листинг 2. Структура документа в коллекции *processed_urls*

В документах коллекции *processed_urls* хранится информация: тип

"*_id*" : *ObjectId*,
"*url*" : *string*,
"*text*" : *string*,
"*positions*" : *array*,
"*d_search*" : *date*

Листинг 3. Структура документа в коллекции *search_results*

контента и его размер, веб-сервер, на котором работает веб-страница, идентификатор ответа веб-сервера, по которому можно определить доступность страницы, рефералы и количество ссылок, найденных на этой странице.

Для хранения результатов поиска используется коллекция *search_results*. Структура документов в этой коллекции приведена в листинге 3.

В документах коллекции *search_results* хранится адрес веб-страницы, на которой проводился поиск, текст поиска и позиции, на которых этот текст был найден.

Рабочий узел поискового робота

Рабочий узел реализован в виде модуля *web_crawler_slave*. Этот модуль экспортирует функции *process* для запуска процесса, который обрабатывает веб-страницы и *add_pending_url* для добавления новой ссылки в коллекцию *pending_urls* и ее дальнейшей обработки.

Функция *process* имеет *обязательный* параметр *URL* и *необязательный – Referral_URL*, т.е. список рефералов. Она реализована по следующему алгоритму.

Функция получает на вход веб-страницу, которую требуется обработать. Подключается к базе данных. Получает из базы дополнительные данные о текущем задании. Изменяет его статус на *processing*. Выполняет нормализацию адреса. Опрашивает веб-страницу по данному адресу, отделяет полученные заголовки и тело самой страницы. При необходимости выполняет дополнительную обработку веб-страницы, например, поиск информации. Находит новые ссылки и записывает их в базу данных. Добавляет информацию о текущей странице в коллекцию *processed_urls*. Удаляет веб-страницу коллекции *pending_urls* и завершает работу.

Для взаимодействия *Erlang* с *MongoDB* используется библиотека *mongodb-erlang*, которая находится в свободном доступе на *github* [14]. В ней реализованы все функции, необходимые для работы с этой СУБД. В листинге 4 приведен код, выполняющий подключение к базе данных и запрашивающий документы из коллекции *pending_urls* с определенным *url*.

```
application:start (mongodb),
case mongo:connect (?MongoDBHost) of
  {ok, MongoConn} –> {ok, Row_pending}=mongo:
   do(safe, master, MongoConn, crawler, fun()–>mongo:
    find_one (pending_urls,{url, URL})
              end);
  {error, Reason} → exit(normal)
 end.
```

Листинг 4. Подключение и запрос к *MongoDB*

Во избежание незапланированной повторной обработки страницы была реализована функция нормализации адреса. Ее код приведен в листинге 5. Функция использует вспомогательные функции, приведенные в листинге 6.

```
normalize_url(URL)–>
  URL_string=bitstring_to_list(URL),
  {Scheme, UserInfo, Host, Port, Path, Query}=http_uri:
  parse(URL_string),
```

```
    list_to_binary(lists:append([atom_to_list(Scheme),
    "://",get_right_userinfo(UserInfo),string:to_lower
      (Host),
    get_right_port(Scheme,Port),get_right_path(Path),
     Query])).
```

Листинг 5. Функция для нормализации веб-адресов

```
is_default_port(http,80) –> true;
is_default_port(https,443) –> true;
is_default_port(Scheme,Port)–>false.

get_right_port(Scheme,Port) –>
    case is_default_port(Scheme, Port) of
            'false' –> string:concat(":", Port);
            'true' –> []
    end.

get_right_userinfo([])–> [];
get_right_userinfo(UserInfo)–>string:concat(UserInfo,
"@").

path_to_upper(Path,[]) –> Path;
path_to_upper(Path,[Pos|Rest]) –>
    lists:append([
            lists:sublist(Path, element(1,lists:last(Pos))-1),
              string:to_upper(lists:sublist(Path,
  element(1,lists:last(Pos)), element(2,lists:last(Pos)))),
                    path_to_upper(lists:sublist(Path,
  element(1,lists:last(Pos))
      +element(2,lists:last(Pos)),length(Path)),Rest)
]).

get_right_path(Path) –>
    {ok, Reg_replase }=re:compile("\\.\\./|\\./"),
    New_path=re:replace(Path,Reg_replase ,"",[global,
      {return,list}]),
    {ok,Regexp}=re:compile("%[0-9abcdef][0-9abcdef]"),
    case re:run(New_path,Regexp ,[global]) of
          {match, Matches} –> path_to_upper
           (New_path,Matches);
            nomatch –> New_path
    end.
```

Листинг 6. Вспомогательные функции для нормализации веб-адресов

Процесс нормализации состоит из: приведения адреса узла к нижнему регистру; отвержения номера порта при использовании его по умолчанию; приведения закодированных символов в адресе к верхнему регистру; удаления лишних точек.

Для работы с веб-средой используется библиотека *inets*. В частности, для загрузки веб-страницы используется функция request из модуля *httpc*, а для обработки ссылки – функция parse из модуля *http_uri*. Пример кода для получения страницы приведен в листинге 7.

```
inets:start(),
  case httpc:request(URL) of
      {ok, {_Status, H, HTML}}→process_body(HTML);
      {error, Reason} → exit(normal)
  end.
```

Листинг 7. Загрузка веб страницы

```
extract_urls(_,[],_,_,_)-> [];
extract_urls(Body,[Pos|Positions],Conn,Referral_URL,
      Operation)->
    URL=normalize_url(binary_part(Body,element(1,
       lists:last(Pos)),
            element(2,lists:last(Pos)))),
 add_pending_url(Conn,URL,Referral_URL,Operation),
 extract_urls(Body,Positions,Conn,Referral_URL,Operation).
find_urls(Body,Referral_URL,Conn,Operation)->
      {ok,Regexp}=re:compile("<a[^\>]+href=[\"\']([^\"\
        ']+)[\"\']"),
      case re:run(Body,Regexp,[global]) of
            {match, Matches} ->
                  extract_urls(Body,Matches,
                   Conn,Referral_URL,Operation),
                  length(Matches);
            nomatch -> 0
      end.
```

Листинг 8. Функции для получения веб-адресов со страницы

После получения страницы в ней ищут новые адреса.

Для поиска веб-адресов и их выделения на странице реализованы функции *find_urls* и *extract_urls*. Они приведены в листинге 8.

Ссылки, найденные на странице не сразу добавляются в базу данных. Сначала они проходят валидацию, чтобы определить, может ли этот адрес быть обработанным. Затем проверяется наличие этого адреса в коллекции *processed_urls*. Если его там нет (он еще не обрабатывался), адрес добавляется к коллекции *pending_urls*. Если этот адрес уже обрабатывался, согласно политике повторной обработки, определяется его возраст, т.е. время, прошедшее с момента его обработки. Если время превышает 60 с, адрес добавляется к *pending_urls* для повторной обработки. В противном случае текущий реферал этого адреса добавляется к документу в коллекции *processed_urls*. Код описанной функции приведен в листинге 9.

```
add_pending_url(Conn,URL,Referral_URL,Operation)->
 case http_uri:parse(binary_to_list(URL)) of
  {error, no_scheme} ->
   {Scheme, UserInfo, Host, Port, Path, Query}=
   http_uri:parse(binary_to_list(Referral_URL)),
   New_URL=lists:append([atom_to_list(Scheme),"://",
  get_right_userinfo(UserInfo),Host,get_right_port(Sche-
       me,Port),URL]),
add_pending_url(Conn,New_URL,Referral_URL,Operation);
  {error, _} -> {ok};
  {Scheme, UserInfo, Host, Port, Path, Query}->
  {_, _, Referral_Host, _, _, _}=http_uri:parse(binary_to_
     list(Referral_URL)),
  if Host= =Referral_Host ->
   Norm_URL=normalize_url(URL),
   {ok,Row_processed}=mongo:do(safe, master, Conn,
     ?DB, fun() ->
    mongo:find_one(?COLLECTION2,{'_id', Norm_URL,
      operation,Operation}) end),
   if tuple_size(Row_processed)>0 ->
Row_processed_list=list2tupleslist(tuple_to_list(element
      (1,Row_processed))),
D_processed=atom_to_list(element(2,lists:keyfind(d_pro
      cessed,1,
     Row_processed_list))),
Age=calendar:datetime_to_gregorian_seconds(calendar:
      local_time())-
     stdate_to_sec(D_processed);
    true -> Age=1000
   end,
   if Age<?REPROCESS_INTERVAL ->
    add_referral_to_processed_url(Conn,
    Norm_URL,Referral_URL,Operation);
    true->
     {ok,Row_pending}=mongo:do(safe, master, Conn,
     ?DB, fun() ->
     mongo:find_one(?COLLECTION1,{url,
      Norm_URL,status,pending,operation,Operation})
     end),
     if tuple_size(Row_pending)>0 ->
Row_pending_list=list2tupleslist(tuple_to_list(element
       (1, Row_pending))),
      mongo:do (safe, master, Conn, ?DB, fun() ->
       mongo:repsert (?COLLECTION1, {url, Norm_URL
        status,pending},
      {url, Norm_URL,referral_list,lists: umerge(element(2,
       lists:keyfind(referral_list,1,Row_pending_list)),
     [Referral_URL]),
status,pending,d_change,stdate(),operation,Operation})
    end);
true -> mongo:do (safe, master, Conn, ?DB, fun() ->
  mongo:repsert (?COLLECTION1, {url, Norm_URL,
   status,pending},
    {url, Norm_URL,referral_list,[Referral_URL],status,
    pending,d_change,stdate(),
      operation,Operation}) end)
   end, {ok} end; true -> {ok} end end.
```

Листинг 9. Функция добавления новой ссылки для дальнейшей обработки

Главный блок поискового робота

Блок состоит из нескольких модулей.

Модуль *web_crawler_app* базируется на *application* и состоит из функций *start* и *stop*. Он контролирует правильный запуск и остановку ПР. Код модуля приведен в листинге 10.

```
-module(web_crawler_app).
-behaviour(application).
-export([start/2, stop/1]).

start(_Type, _Args) –>
      web_crawler_sup:start_link().
stop(_State) –>
      ok.
```

Листинг 10. Модуль *web_crawler_app*

```
-module(web_crawler_sup).
-behaviour(supervisor).
-export([start_link/0]).
-export([init/1]).
start_link() –>
      supervisor:start_link({globa,supervisor},
      web_crawler_sup, []).
init(_Args) –>
      pool:start(crawler_pool),
      {ok, {{one_for_one, 1, 60},
      [{web_crawler_master, {web_crawler_master,
        start_link, []},
            permanent, brutal_kill, worker,
             [web_crawler_master]}]}}.
```

Листинг 11. Модуль *web_crawler_sup*

Модуль *web_crawler_sup* базируется на поведении *supervisor*. Его основная задача – запуск главного процесса поискового робота и контроль за его работой, т.е. перезапуск в случае отказа. Модуль состоит из функции *start_link*, который запускает *web_crawler_app*, и функции *init*, в которой прописано, как запускать главный поток, и алгоритм его перезапуска. Коды модуля приведены в листинге 11.

Последний и главный модуль – *web_crawler_master*. Он базируется на поведении *gen_server*. Его главная задача – проверка наличия веб-адресов, ожидающих обработки, и распределение задач на обработку между рабочими узлами.

После инициализации модуля, включается функция *process_urls*, уходящая в хвостовую рекурсию и вызываемая. пока работа не будет остановлена. Эта функция подключается к базе данных, выбирает все ссылки из коллекции *pending_urls* и вызывает функцию *distribute_urls*, которая в свою очередь создает процессы для их обработки на рабочих узлах. Код функций приведен в листинге 12.

```
process_urls(MongoConn,Log,0) –>
      exit(normal);
process_urls(MongoConn, Log, Times) –>
      logger:write(Log,"Processing pending urls.
        Times=~p~n",[Times]),
      {ok,Result}=mongo:do(safe, master, MongoConn,
        ?DB, fun() →
      mongo:find(?COLLECTION1,{status,pending})end),
      Rows=mongo:rest(Result),
      logger:write(Log, "Found ~p urls. Starting distribu
        tion.~n",
                    [length(Rows)]),
      distribute_urls(Rows),
      if is_integer(Times) –> T=Times -1;
        true –> T= Times
      end,
      process_urls(MongoConn,Log,T).

distribute_urls([])–> {ok};
distribute_urls([Row| Rows]) –>
      pool:pspawn(web_crawler_slave,process,
      [element(4, Row),element(6, Row)]),
      distribute_urls(Rows).
```

Листинг 12. Функции *process_urls* и *distribute_urls*

Благодаря использованию поведения *gen_server*, модуль *web_crawler_master* может принимать синхронные и асинхронные запросы. С использованием асинхронного запроса реализована внешняя функция *add_url*, определяющая новый адрес для обработки ПР.

Также реализована функция для поиска определенного текста на веб-страницах. Она использует функцію *add_url* и добавляет новый адрес в коллекцию *pending_urls*. Но в этом случае в поле *operation* отмечается, что это поиск текста. Коды этих функций приведены в листинге 13.

Механизм работы в распределенном режиме. Наш ПР может работать в многопоточном режиме на одном компьютере, но тогда

```
add_url(URL) –>
  gen_server:cast(web_crawler_master,{add_url,URL, {}}).
add_url(URL, Operation) –>
  gen_server:cast(web_crawler_master, {add_url, URL,
    Operation }).

search_text(URL,Text) –>
      add_url(URL, {search,list_to_binary(Text)}).

handle_cast({add_url,URL,Operation},{MongoConn,Fd})–>
      web_crawler_slave:add_pending_url(MongoConn,
        list_to_binary(URL),
                                    [],Operation),
  {noreply,{MongoConn,Fd}}.
```

Листинг 13. Функции *add_url* и *search_text*

он должен был быть распределенным. Для этого были использованы модуль *pool*, входящий в состав *Erlang*. Он позволяет объединять рабочие узлы в пул, после чего главный узел будет указывать пулу на создание нового процесса, а на каком именно узле его создавать будет решать сам контроллер пула. Этим достигается балансировка нагрузок, так как новый процесс всегда создается на наименее загруженном узле, а также постоянно поддерживается актуальная информация об имеющихся в пуле узлах. Следует отметить, что добавлять новые узлы можно без остановки работы системы.

Для запуска пула необходимо выполнить следующие шаги [15]. Подготовить набор компьютеров с одинаковой средой *Erlang*, один из которых будет главным. Послать на все компьютеры программный код, который будет запускаться. На главном узле создать файл *.Hosts.erlang*, в котором прописать названия всех рабочих узлов, по которым к ним можно присоединиться. Создать на всех узлах одинаковый файл *.Erlang.cookie*, в котором должна быть записана строка текста, общая для всех узлов, чтобы компьютеры одного пула «видели» друг друга. Настроить на главном узле беспарольный доступ через *ssh* ко всем рабочим узлам. Запустить на главном узле именуемую среду *Erlang*. Это достигается командой *erl-sname <имя узла>*.

Затем инициализируем пул командой *pool: start (<имя пула>)*, а процессы в нем создаются командой *pspawn (<имя модуля>, <имя функции>, <параметры функции>)*. Таким образом

была реализована работа с пулом в поисковом роботе и он стал распределенным.

Заключение. Описано построение ПР со следующими функциями: построение структуры веб-ресурса, мониторинг доступности его страниц и поиск теста. Он имеет клиент-серверную архитектуру и работает с СУБД *MongoDB*.

В качестве дальнейшего развития системы было бы полезно осуществить удаленную загрузку кода. Чтобы запустить процесс на рабочем узле, необходим код. В *Erlang* есть модуль *erl_boot_server*, который при правильной настройке позволяет загружать код из главного узла в момент запуска среды на рабочем узле, что устраняет необходимость постоянной поддержки актуальной версии кода на всех узлах.

Для ускорения работы с базой данных можно использовать репликацию на несколько узлов [16]. В *Erlang*, с помощью конструкции *replset*, можно балансировать запросы между разными узлами, что ускорило бы операции чтения и дало значительный прирост быстродействия на большом количестве рабочих процессов.

1. *Глибовец Н.Н.*, *Гороховский С.С.* Проектирование инструментария сетевого программного продукта. Современное состояние и перспективы развития // Проблеми програмування. – 2010. – № 2–3. – С. 102–107.
2. *citeseerx*.ist.psu.edu/viewdoc/download?doi=10.**1**.**1**. 202. 692...
3. *www*.cs.kent.ac.uk/projects/ofa/.../**tutorial**.p...
4. *https*://wiki.cse.unsw.edu/cs4181/12s2/Presentation%20 Topicsr
5. *Глибовець А.М.*, *Шабінський А.С.* Один підхід до побудови інтелектуальної пошукової системи // Наук. зап. НаУКМА. Комп'ютерні науки. – 2010. –**112**. – С. 26–30.
6. *ilpubs*.stanford.edu:8090/604/1/**2003**-44.pdf
7. *www*.**erlang**. org/.../**erlang**-book-part1.pdf
8. *http*://en.wikipedia.org/wiki/Web_crawler
9. *Глибовец Н.Н.*, *Глибовец А.Н.*, *Шабинский А.С.* Применение онтологий и методов текстового анализа при создании интеллектуальных поисковых систем // Проблемы управления и информатики. – 2011. – № 6. – С. 95–102.
10. *Menczer F.* ARACHNID: Adaptive Retrieval Agents Choosing Heuristic Neighborhoods for Information Discovery / D. Fisher, ed., Machine Learning: Proc. of the 14th Int. Conf. (ICML97), 1997
11. *http*://en.wikipedia.org/wiki/Robots_Exclusion_Standard
12. *http*://en.wikipedia.org/wiki/Distributed_web_crawling
13. *http*://en.wikipedia.org/wiki/MongoDB
14. *stackoverflow*.com/.../**mongodb-erlang-erla**...
15. *http*://habrahabr.ru/post/114663/
16. *http*://express-js.ru/mongo-book/Glava-7-Proizvoditelnost-i-instrumentarij.html

Поступила 18.10.2012

Тел. для справок: +38 044 463-6985 (Киев)
E-mail: *glib@ukma.kiev.ua*