J Netw Syst Manage (2016) 24:34–56
DOI 10.1007/s10922-014-9338-0

Distributed Algorithms for a Replication Problem of Popular Network Data

Xiaofeng Jiang · Jun Li · Hongsheng Xi

Received: 30 December 2012 / Revised: 10 November 2014 / Accepted: 11 December 2014 /
Published online: 21 December 2014

Abstract The replication of popular data objects can effectively reduce the access time and bandwidth requirements of network services. We study the replication problem in the model of distributed replication groups and propose two distributed algorithms: an approximation optimal replication algorithm, which is an asynchronous distributed algorithm as it takes more time to be completed. However its performance approaches the optimal algorithm, and a fast replication algorithm that is very suitable as the initial algorithm of the approximation optimal algorithm. We give a proof of the complexity of the algorithms, and show that the time and communication complexities of the algorithms are polynomial with respect to the number of objects and the maximum storage capacities of the servers. Finally, simulation experiments are performed to investigate the performance of the algorithms, and the results show that the two algorithms can effectively solve the replication problem.

Keywords Replication of popular data · Access cost · Distributed replication group · Asynchronous distributed algorithm

1 Introduction

The access time and bandwidth requirements are always two important issues of network application services. With the rapid growth of network data and the

X. Jiang (✉) · J. Li · H. Xi
Department of Automation, University of Science and Technology of China, Hefei 230021, China
e-mail: jxf@ustc.edu.cn; jxf@mail.ustc.edu.cn

J. Li
e-mail: Ljun@ustc.edu.cn

H. Xi
e-mail: xihs@ustc.edu.cn

prevalence of distributed systems, recent efforts on improving network services have paid more attention to replication algorithms of network resources. A server maintains a copy of popular network data objects locally according to the replication algorithm, and then provide it to local users or other servers that are close to the server. A good replication algorithm can efficiently use the storage capacity to replicate network data objects and acquire the best access performance. A commonly considered model to study such a problem is the distributed replication group.

A distributed replication group consists of a number of servers that provide storage capacities to replicate network data objects. The requests of a user are processed by a local server. If the local server has stored the required data objects, the requests can be responded to in a very short period of time. Otherwise, the local server should retrieve the storage resources of other servers in the same group. If the required data objects are found, the requests can be responded to within a longer period of time. In the case that no required data objects are fetched in the group, the local server must contact the original server that lays outside of the group to get the required data, the response time will be very long.

Consequently, the replication problem can be defined as follows. Each server receives the requests for network data objects in accordance with certain rates, and the servers replicate some data objects to serve the requests that come from local users and nonlocal users within the group. The replication algorithm should efficiently use the storage resources and maximize the total performance of all the servers in the group. Two distributed replication algorithms are proposed to solve the problem in our study: An approximation optimal replication algorithm (AORA) and a fast replication algorithm (FRA). AORA is an asynchronous distributed algorithm with shared memory, it will spend more time to complete the configuration, but its results will approach those of the optimal algorithm. Then we will give the detailed proofs of the time and communication complexities of the algorithm after its description. The FRA is a three-step optimization algorithm that can complete the configuration of the servers' storage resources in a short period of time. It is very suitable as the initial algorithm of the AORA for the reason that it can make the configuration close to the optimal state and greatly reduce the running time of the approximation optimal algorithms. The FRA can also be suitable for high real-time requirements due to its low latency. At last, we will compare the various performances of the AORA and FRA with the algorithms that have been proposed.

Some replication algorithms have been proposed to solve the replication problems in different environments. Leff [1] classifies the algorithms for the model of distributed replication group as three kinds of algorithms: optimal algorithms, distributed algorithms and isolationist algorithms, and then tests their performances by simulations. An optimal algorithm shares all the access patterns at every server and the replication decisions are completely coordinated, so at every moment, the best decisions are made. The distributed algorithm doesn't make decisions in a completely coordinated fashion but will consider the information of other servers. The isolationist algorithm ignores all the decisions made by other servers when a server replicates data objects. The optimal algorithms require a lot of computing

resources and time, and the performance derived by isolationist algorithms can't meet the requirements of systems. Therefore a distributed algorithm is thus more appropriate, the results of simulations show us that the distributed algorithm is a more appropriate solution.

Zaman [2] and Laoutaris [3] also consider the model of the distributed replication group and propose two distributed algorithms: distributed greedy replication (DGR) and two-step local search (TSLS). The performances of these two algorithms will be compared with the algorithm that we propose in the section of experiments. The time complexity of the TSLS is low, but it doesn't approach the performance of the optimal algorithm. The DGR is an approximation algorithm, but the time complexity is very high. In our work, the time complexity of the AORA is proved that it has been reduced by the times of the number of servers, and under some conditions, the performance can remain the same. Khan [4] proposes the $A\epsilon$-star algorithm that is used to replace the optimal algorithm for comparison in the section of experiments. The other replication algorithms for a variety of systems are considered in [5–23]. The studies of [5–14] are used to minimize the data access cost and [15–23] consider the algorithms that can improve the system reliability. The performance analysis is a key method to validate the proposed algorithms. The studies of [24–27] use the Markov chain to model the network transmission procedure and do the performance evaluation, they can provide great help to the simulation of the proposed algorithms. The studies of [28–30] provide more ideas for the later research in the distributed and cloud computing systems.

2 Problem Formulation

We consider that the distributed replication group consists of M servers, and there are N data objects of unit size that will be replicated in the group. Let $s_m, 1 \leq m \leq M$ and $o_n, 1 \leq n \leq N$ denote the mth server and the nth data object respectively, and s_m is supposed to have a storage capacity of c_m. The request rates of objects are described as a $M \times N$ matrix r, and its member $r_{m,n}$ denote the number of requests for o_n that arrive at s_m in a unit time.

Since there are many notations in the definition of the problem, we want to give a summary list to make the notations clear in Table 1.

We assume that a server can access the data object with the access cost t_l, if the object is stored locally. Else, if the object can be retrieved in other servers of the group, the access cost becomes t_r. Otherwise, if the object can only be found in the original server which is outside of the group, the access cost is t_s. In the actual model of [1], t_l, t_r and t_s are given values of tenths of millisecond, milliseconds and tens of milliseconds, respectively. We can see that t_s is much larger than t_r and t_l in the actual background. The collaboration of the storage resources of servers in the same group becomes meaningful when the access cost of the original server is large. Here, we adopt the model, which has been defined in the study of [2].

Each server knows the copies of objects stored locally. The copies of all servers are described as M vectors $\{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_M\}$, and one member of $\mathbf{X}_m$ is given by

Table 1 Summary notations

s_m	Server m
o_n	Data object n
c_m	Storage capacity of s_m
$r_{m,n}$	Request rate for o_n at s_m
t_l	Access cost of the local server
t_r	Access cost of the group
t_s	Access cost of the original server
$X_{m,n}$	Number of o_n at s_m
AC	Total access cost of the group
ACO	Maximum access cost of the group
PG	Performance gain of the group
$pg^+_{m,n}$	Performance gain of s_m for storing o_n
$pg^-_{m,n}$	Performance gain of s_m for deleting o_n
$tpg^+_{m,n}$	Performance gain of the group for storing o_n
$tpg^-_{m,n}$	Performance gain of the group for deleting o_n

$$X_{m,n} = \begin{cases} 1 & \text{if } s_m \text{ stored a copy of } o_n \\ 0 & \text{otherwise} \end{cases}.$$

Then, the total access cost of the group AC [2] can be given by

$$\sum_{X_{m,n}=1} r_{m,n}t_l + \sum_{\substack{X_{m,n}=0 \\ \sum_{m=1}^{M} X_{m,n} \geq 1}} r_{m,n}t_r + \sum_{\sum_{m=1}^{M} X_{m,n}=0} r_{m,n}t_s \tag{1}$$

subject to

$$\sum_{n=1}^{N} X_{m,n} \leq c_m, 1 \leq m \leq M.$$

The first term of (1) represents the total access cost when the required objects are stored locally. The second term represents that the required objects are stored in the group, but they are not stored locally. The third term represents that the required objects can only be found in the original server. The replication algorithm should make its best effort to minimize AC. From the opposite side, when the distributed replication group doesn't replicate any data objects in the servers, all the servers have to access the original servers to serve the users, the access cost ACO [2] is maximum, and it can be given by

$$ACO = \sum_{m=1,n=1}^{M,N} r_{m,n}t_s.$$

Consequently, we can denote the performance gain PG [2] of a replication algorithm for the group as

$$PG = ACO - AC.$$

By transforming the equation above, we can obtain that

$$PG = \sum_{X_{m,n}=1} r_{m,n}(t_s - t_l) + \sum_{\substack{X_{m,n}=0 \\ \sum_{m=1}^{M} X_{m,n} \geq 1}} r_{m,n}(t_s - t_r) \tag{2}$$

The first term of (2) represents the performance gain when the required data objects are stored locally, and the second term represents the performance gain while the required data objects are stored in the group. So the higher the value of PG, the better the replication algorithm.

To describe the algorithms, two important parameters $pg^{+}_{m,n}$ and $pg^{-}_{m,n}$ must be defined. Let $pg^{+}_{m,n}$ denote the performance gain of s_m when it stores a copy of o_n locally, and $pg^{+}_{m,n} \geq 0$. Let $pg^{-}_{m,n}$ denote the performance gain of s_m when it deletes a copy of o_n that stored locally, and $pg^{-}_{m,n} \leq 0$. First, to simplify the expressions of the parameters, some restrictions are given simplified notations in Table 2.

Then, the values of the parameters can be obtained by the following equations.

$$pg^{+}_{m,n} = \begin{cases} r_{m,n}(t_r - t_l) & case^{+}_{1} \\ r_{m,n}(t_s - t_l) & case^{+}_{2} \\ 0 & case^{+}_{3} \end{cases} \tag{3}$$

$$pg^{-}_{m,n} = \begin{cases} -r_{m,n}(t_r - t_l) & case^{-}_{1} \\ -r_{m,n}(t_s - t_l) & case^{-}_{2} \\ 0 & case^{-}_{3} \end{cases} \tag{4}$$

With the definition of $pg^{+}_{m,n}$ and $pg^{-}_{m,n}$, we can get the impact of the replication and deletion operations of s_m on the total performance gain of the group. Let $tpg^{+}_{m,n}$ denote the total performance gain of the group when s_m stores a copy of o_n locally, and $tpg^{+}_{m,n} \geq 0$. Let $tpg^{-}_{m,n}$ denote the total performance gain of the group when s_m deletes a copy of o_n that stored locally, and $tpg^{-}_{m,n} \leq 0$. Their values can be obtained by the following equations.

Table 2 Simplified notations of restrictions

Case	Restrictions
$case^{+}_{1}$	$X_{m,n} = 0, \sum_{m=1}^{M} X_{m,n} \geq 1$
$case^{+}_{2}$	$X_{m,n} = 0, \sum_{m=1}^{M} X_{m,n} = 0$
$case^{+}_{3}$	$X_{m,n} = 1$
$case^{-}_{1}$	$X_{m,n} = 1, \sum_{m=1}^{M} X_{m,n} \geq 2$
$case^{-}_{2}$	$X_{m,n} = 1, \sum_{m=1}^{M} X_{m,n} = 1$
$case^{-}_{3}$	$X_{m,n} = 0$

$$tpg^{+}_{m,n} = \begin{cases} r_{m,n}(t_r - t_l) & case^{+}_{1} \\ \sum_{i=1}^{M} r_{i,n}(t_s - t_r) + r_{m,n}(t_r - t_l) & case^{+}_{2} \\ 0 & case^{+}_{3} \end{cases} \quad (5)$$

$$tpg^{-}_{m,n} = \begin{cases} -r_{m,n}(t_r - t_l) & case^{-}_{1} \\ -\sum_{i=1}^{M} r_{i,n}(t_s - t_r) - r_{m,n}(t_r - t_l) & case^{-}_{2} \\ 0 & case^{-}_{3} \end{cases} \quad (6)$$

Under $case^{\pm}_{1}$ and $case^{\pm}_{3}$, the expressions of $tpg^{\pm}_{m,n}$ can be obtained from the definition of PG. Under $case^{+}_{2}$, there is not any copy of o_n in the group. If the replication operation of o_n happens in s_m, the other servers in the group can access o_n in s_m rather than the original server, this affects the performance of the group, so we get the expression in Eq. (5). Under $case^{-}_{2}$, there is only one copy of object o_n in the group. The deletion operation of o_n also affects the performance of the other servers, so we can get the expression in Eq. (6).

3 Approximation Optimal Replication Algorithm

The AORA is an asynchronous distributed algorithm with the shared memory, it essentially makes the servers in a group perform the non-critical operations asynchronously. Here, the non-critical operation denotes the operation that doesn't affect the performance gains of the other servers, such as the operation under $case^{\pm}_{1}$ or $case^{\pm}_{3}$. The critical operation denotes the operation that affects the performance gains of the other servers, such as the operation under $case^{\pm}_{2}$.

The shared memory denotes the shared variables that can be writable and readable by all servers. The group will be locked and all servers stop the operations if a critical operation arrives, this criterion can guarantee the total performance gain of the group. Since the number of non-critical operations is much larger than the critical operations', the AORA can greatly reduce the running time of the replication algorithm.

To describe the algorithm, we first define the vector $\mathbf{s}^c = (s^c_1, \ldots, s^c_N)$ where s^c_n is denoted by $\sum_{m=1}^{M} X_{m,n}$. Let n_d and n_r denote the sequence numbers of two objects that will be deleted and replicated in a server, respectively. Let n^i_d denote the sequence number of the object that has been replicated in a server for $1 \le i \le c_m$. Let n^j_r denote the sequence number of the object that has not been replicated in a server for $1 \le j \le N - c_m$. Let t_{wait} denote the total sum of the maximum processing time of actions $try{-}update_m, update_m, crit{-}update_m, send_m$ and $receive_m$, which will be defined in the following algorithm. Meanwhile, s_m should maintain a replication vector X_m. With these definitions, the description of the AORA is given in Algorithm 1.

Algorithm 1. AORA
Shared variables:

$s_n^c \in \{0, 1, ..., M\}$, initially generated by initial algorithm $s_n^c = \sum_{m=1}^{M} X_{m,n}$, writable and readable by all servers, for every $n \in \{1, ..., N\}$.
$turn \in \{0, 1, ..., M\}$, initially 0, writable and readable by all servers. If $turn = 0$, each server can replicate and delete the copies of objects, and modify the values of $\mathbf{s}^c$. If $turn = m$, only s_m can perform operations and modify $\mathbf{s}^c$.

Actions of s_m:

$try - update_m$ examines whether the copies of s_m should be updated.
$update_m$ examines whether the group has been locked, and if not, updates the copies of s_m.
$crit{-}update_m$ updates the copies of s_m, if the group has been locked.
$wait_m$ examines whether the waiting time has expired.
$finish_m$ stops the algorithm.
$receive_m$ receives the message.
$send_m$ broadcasts the message.

States of s_m:

$pc \in \{try{-}update, update, crit{-}update, wait, finish\}$, initially $try - update$.

Transitions of s_m:
$try - update_m$:

Precondition:
$pc := try - update$;
Effect:
for every $n \in \{1, ..., N\}$
if $X_{m,n} = 1$, then calculate $tpg^-_{m,n}$;
else if $X_{m,n} = 0$, then calculate $tpg^+_{m,n}$;
end if
end
choose:
$-tpg^-_{m,n_d} = min\{-tpg_{m,n_d^1}, ..., -tpg_{m,n_d^{c_m}}\}$;
choose:
$tpg^+_{m,n_r} = max\{tpg_{m,n_r^1}, ..., tpg_{m,n_r^{N-c_m}}\}$;
if $-tpg^-_{m,n_d} < tpg^+_{m,n_r}$, then $pc = update$;
else $pc = wait$;
end if

$update_m$:

Precondition:
$pc := update$;

Effect:
if $turn = 0$, then
if $s^c_{n_d} = 1$ or $s^c_{n_r} = 0$, then
$turn = m$; $pc = crit - update$;
else if $s^c_{n_d} > 1$, then
$s^c_{n_d} = s^c_{n_d} - 1$; $s^c_{n_r} = s^c_{n_r} + 1$;
$X_{m,n_d} = 0$; $X_{m,n_r} = 1$;
end if

else if $turn = m$, then $pc = crit - update$;
else $pc = wait$;
end if
$sendbuff =$ {replication placement change};
$pc = try - update$;

$crit - update_m$:

Precondition:
$pc := crit - update$;
Effect:
if $turn = m$, then
$s^c_{n_d} = s^c_{n_d} - 1$; $s^c_{n_r} = s^c_{n_r} + 1$;
$turn = 0$;
$X_{m,n_d} = 0$; $X_{m,n_r} = 1$;
end if
$sendbuff =$ {replication placement change};
$pc = try - update$;

$wait_m$:

Precondition:
$pc := wait$;
Effect:
while (wait time t_{wait})
if $receivebuff \neq NULL$, then
$receivebuff = NULL$;
$pc = try - update$;
end if
end
$pc = finish$;

$finish_m$:

Precondition:
$pc := finish$;
Effect:
stop AORA algorithm at server s_m;

$send_m$:

Precondition:
$sendbuff \neq NULL$;
Effect:
broadcast the message of sendbuff to all servers;
$sendbuff = NULL$;

$receive_m(message)$:

Precondition:
receive a message;
Effect:
add the message to receivebuff;

Since the detailed description of the AORA is complex and unreadable, we summarize the state transition process in Fig. 1 to give a concise description. Initially, the state is *try-update* in a server. Then, if the data allocation of this server is globally optimal, the state goes to *wait*, else the state goes to *update*. The server should first check whether the system has been locked when its state is *update*. If the system has been locked, the server's state should go to *wait*, else the server checks whether the system needs to be locked in order to perform the update operation.

According to the result, the server's state will go to *crit-update* or return *try-update*. We should note that the state of a server may leave *wait* when it is motivated by an update message. The detailed text description is given as follows.

For the server s_m, first, it will set the state as *try-update* which leads to *try-update*$_m$. In *try-update*$_m$, s_m calculates the minimum of the performance degradation of the group when a copy of o_{n_d} is deleted. Then the maximum of the group's performance improvement will also be calculated when a copy of o_{n_r} will be replicated. To improve the the group's performance, s_m can decide whether to replace n_d with n_r by comparing these two values. If the server tries to replace the copies, the state goes to *update*, and the server performs *update*$_m$, else the state goes to *wait*.

In *update*$_m$, s_m first checks *turn* to see if $turn = 0$. If not, and if the current value of *turn* is not m, the state goes to *wait*. If $turn = 0, s_m$ checks $s^c_{n_d}$ and $s^c_{n_r}$. If $s^c_{n_d} > 1$ and $s^c_{n_r} > 0$, the replication and deletion operations of s_m will not affect the performances of the other servers, the group doesn't need to be locked. When the replacement of the copies of o_{n_d} and o_{n_r} is completed, the *sendbuff* is filled with the message {replication placement change} to be broadcast to the other servers. The state will be back to *try-update* for examining that whether the copies of s_m need to be optimized again. If $s^c_{n_d} = 1$, which means that the group has only one copy of o_{n_d}, the deletion operation of s_m will affect the performances of the other servers. If $s^c_{n_r} = 0$, which means that the group doesn't have any copy of o_{n_r}, the replication operation of s_m will affect the performances of the other servers, the value of *turn* will be set as m to lock the group, and the state goes to *crit-update*.

In $crit - update_m, s_m$ will first check *turn* to see if $turn = m$. If it is, when the copies of o_{n_d} and o_{n_r} are completed, *turn* is set as 0 to unlock the group, the *sendbuff* is filled with the message {replication placement change} to be broadcast to other servers. Then, the state will be back to *try-update* to examine that whether the copies of s_m needs to be optimized again.

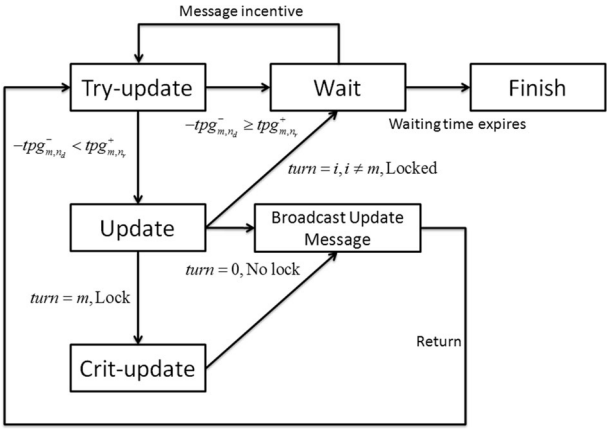

Fig. 1 State transition process of AORA

In $wait_m, s_m$ repeatedly checks *receivebuff* to see if *receivebuff* = *NULL* until the waiting time exceeds t_{wait}. If not, which means the copies in the group has changed, and this affects the configuration in s_m, the state should go to *try-update*. Once the waiting time expires, the state will go to *finish*, and the AORA will be stopped. When the group stays on state *finish*, the AORA is completed.

4 Complexity of the AORA

We analyze the time and communication complexities of the AORA in this section. The time complexity is determined by the maximum number of replacement operations that take place in a server, and the communication complexity is determined by the total number of replacement operations that take place in the group. We will give the proofs of these two propositions in the following lemmas. Before getting the further conclusions, we first define a assumption to limit the values of t_s, t_r and t_l.

Assumption 1 t_s, t_r and t_l are such that $t_s - t_r \gg t_r - t_l$ and

$$\frac{\sum_{j=1}^{M} r_{j,n_2}}{r_{i,n_1}} \frac{t_s - t_r}{t_r - t_l} > 1$$

for each $i, n_1, n_2, 1 \leq i \leq M, n_1 \neq n_2, 1 \leq n_1, n_2 \leq N$.

Assumption 1 is always true in the actual network model.

Let C denote the total sum of the storage resources of the group $\sum_{m=1}^{M} c_m$. Let n_{ro} denote the maximum number of replacement operations which take place in a server. Let t_{ro} denote the maximum time of one replacement operation. Then, we can give the following proposition.

Proposition 1 *The time complexity of the AORA is determined by the maximum number of replacement operations that take place in a server.*

We first give a lemma to support Proposition 1.

Lemma 1 *s_m doesn't replace any copy that has been stored to optimize the performance gain when its state goes to finish, for each $m \in \{1, 2, \ldots, M\}$.*

Proof By contradiction. We assumed that s_m still needs to replace some copies when its state went to *finish*. There are two cases that may lead to this result. Case 1: when s_m stays in *finish*, it receives a message. Case 2: when the state of s_m is *try-update*, s_m doesn't transfer its state to *wait*.

In case 1, when s_m is in *finish*, we assume that one server $s_i, i \neq m$ performs the replacement operation and broadcasts a message. Because t_{wait} is larger than the total sum of the processing time of all actions, s_m should be in *wait* when s_i is in *try-update*. Then, we consider the action that makes s_i go to *try-update*, the previous state of s_i should be *update*, *crit-update* or *wait*. Consequently, s_i has broadcast a message or received a message. s_m should receive a message in either case, it can't go to *finish*. A contradiction, case 1 doesn't hold.

In case 2, s_m can transfer state from *try-update* and *update* to *wait*. But when the previous state is *update*, by the code, a server is performing replacement operation and then broadcasts a message, s_m can't go to *finish*. So s_m must transfer state from *try-update* to *wait* before going to *finish*. A contradiction, case 2 doesn't hold.

The lemma is thus true. □

Then the proof of Proposition 1 can be given.

Proof With the proof of Lemma 1 in case 1, we can see that there is not any replacement operation in the group. Consequently, the entire group is in *wait*, all the servers will go to *finish* at most t_{wait} time later. The maximum running time should be less than $n_{ro}(t_{ro} + t_{wait})$. So the maximum running time is determined by n_{ro}. □

4.1 Time Complexity

Let c_{max} denote $max\{c_1, c_2, \ldots, c_M\}$, we can give the time complexity of the AORA.

Theorem 1 *If the initial algorithm replicates the copies of all data objects in the group, the time complexity of the AORA is* $O(c_{max}(t_{ro} + t_{wait}))$ *under Assumption 1.*

We first give some lemmas to support Theorem 1.

Lemma 2 *If one copy of* o_n *replaces one copy of* $o_i, i \neq n$ *in a server* s_m*, this copy of* o_n *will never be replaced if no server goes to crit − update after this event for each* $1 \leq n \leq N, 1 \leq m \leq M$.

Proof By the code, if the deletion operation takes place in $update_m$, the value of *turn* should equal 0, and $s^c_{n_d} > 1$. Consequently, there are at least 2 copies of o_{n_d} in the group. So, when s_m deletes the local copy, the other servers can still access o_{n_d} in the group. Only the access cost of s_m has changed from t_l to t_r. The performance gains of replacement operations that take place in the other servers will not change. If the replication operation takes place in $update_m$, the value of *turn* should equal 0, and $s^c_{n_r} > 0$. Consequently, there has been at least 1 copy of o_{n_r} in the group. Therefore, the other servers can still access o_{n_r} in the group when s_m replicates one copy of o_{n_r} locally. Only the access cost of s_m has changed from t_r to t_l. The performance gains of replacement operations that are taking place in the other servers will not change.

We can see that the deletion or replication operation of one server can't affect the performance of the other servers if the server won't go to *crit-update*. Here, s_m has completed the replacement operation, and goes to *try-update*. Because the performance doesn't change, with the result of the previous action $try\text{-}update_m$, we know that the performance improvement of o_n is maximum in the set of objects that have not been replicated in s_m. In the following replacement operations, if no server goes to *crit-update*, the performance improvement of the set of objects, which have not been replicated in s_m, will always be less than that of o_n. Consequently, o_n will never be replaced. □

Lemma 3 *If the group has the copies of all objects, no server will go to crit-update under Assumption 1.*

Proof By contradiction. We assume that s_m goes to *crit-update* and replaces the copy of o_n with o_i for each $1 \le m \le M$, $1 \le n \le N, i \ne n$, the value of s_n^c is 1. s_m will delete the only copy of o_n, and make s_n^c be 0. Meanwhile, because all the objects have copies in the group, the value of s_i^c must be not less than 1. Because of Assumption 1, we can get the following inequality.

$$tpg_{m,n}^{-} = \sum_{j=1}^{M} r_{j,n}(t_s - t_r) + r_{m,n}(t_r - t_l) > \sum_{j=1}^{M} r_{j,n}(t_s - t_r) > r_{m,i}(t_r - t_l) = tpg_{m,i}^{+} \tag{7}$$

A contradiction, the inequality shows us that the copy of o_i can't replace o_n. So if the group has copies of all the objects, no server will go to state *crit-update* under Assumption 1. □

Then, we can prove Theorem 1.

Proof First, we can get that no server will go to state *crit-update* according to Lemma 3. For each $1 \le m \le M, 1 \le n \le N, i \le n$, because once s_m replaces o_n with o_i, the copy of o_i will never be replaced according to Lemma 2, we can get that s_m can perform at most c_m replacement operations during the running time of the AORA. Consequently, the maximum number of replacement operations of one server is c_{max}. Then, we can get that $n_{ro} = c_{max}$, and the time complexity is $O(c_{max}(t_{ro} + t_{wait}))$ according to Proposition 1. □

4.2 Communication Complexity

Let B_{mes} denote the number of bytes of the broadcast messages, we can give the communication complexity of the AORA.

Theorem 2 *If the initial algorithm replicates the copies of all the objects in the group, the communication complexity of AORA is* $O(Mc_{max}B_{mes})$ *under Assumption* 1.

Proof According to Theorem 1, we can get that the maximum number of replacement operations of one server is c_{max}, so the maximum number of replacement operations of the group is Mc_{max}. Each replacement operation broadcasts a message, so the communication complexity of the AORA algorithm is $O(Mc_{max}B_{mes})$. □

5 A Fast Replication Algorithm

The FRA is used as the initial algorithm of the AORA. The FRA is essentially a 3-step optimization algorithm, the performance of the group is optimized in step 2 and step 3. At step 1, we replicate all the data objects in the chosen servers of the group. As a result, each remaining server in step 2 can access the data objects in the chosen servers, the replication and deletion operations of one server can't affect the other servers. Consequently, the remaining servers will become independent of each other. When each remaining server obtains its best performance, the remaining

servers will obtain the best performance. At step 3, we optimize the performance of the chosen servers whose storage capacity is nearly N. Each chosen server can release some storage resources when some copies of objects can be retrieved in the remaining servers. Like step 2, the released storage resources of the chosen servers become independent of each other, we can make these storage resources utilized with their best performances.

Especially, the FRA can greatly supplement the AORA according to Theorems 1 and 2; it can guarantee that the copies of all objects are replicated in the group, the complexity of the AORA can thus be greatly reduced. The details of the FRA are given in Algorithm 2.

Algorithm 2. FRA

Step 1: We first choose M^c servers, and M^c subjects to:

$$M^c \leq M, \sum_{m=1}^{M^c} c_m \geq N, \sum_{m=1}^{M^c-1} c_m < N$$

Then, we replicate all the data objects to these servers according to the following greedy algorithm.

The values of $\{pg^+_{1,n}, pg^+_{2,n}, \ldots, pg^+_{M^c,n}\}$ are calculated under $case^+_2$ for each $1 \leq n \leq N$. If the remaining storage resources of s_m is 0, or one copy of o_n has been replicated in a server, $pg^+_{m,n}$ should equal 0. o_n will be replicated in the server with the maximum value. When all the objects are replicated in the group, the greedy algorithm is completed.

Step 2: The remaining servers $s_{M^c+1}, \ldots, s_M$ calculate the values of $\{pg^+_{m,1}, pg^+_{m,2}, \ldots, pg^+_{m,N}\}$ under $case^+_1$. s_m should replicate the data objects corresponding to the c_m largest values for each $M^c + 1 \leq m \leq M$.

Step 3: s_m deletes all the copies of objects that can be retrieved in the servers $s_{M^c+1}, \ldots, s_M$ for each $1 \leq m \leq M^c$, and calculates the values of $\{pg^+_{m,1}, pg^+_{m,2}, \ldots, pg^+_{m,N}\}$ under $case^+_1$ and $case^+_3$. Then, s_m should replicate the data objects corresponding to the largest values according to the remaining storage resources.

From the implementation of the FRA, we can see that the optimizations in step 2 and step 3 are distributed computing in each server, so the algorithm will finish in a short period of time. Let t_c and t_r be the maximum times of one calculation operation and one replication operation, respectively. The time complexity of the FRA is $O(Nt_c + c_{max}t_r)$.

6 Numerical Validation

In this section, we perform experiments to test the performance of the FRA algorithm and the AORA algorithm in practice.

6.1 Experimental Environment

The experimental environment is set as a video on demand platform, which is shown in Fig. 2. Such a system has been running for two years in our laboratory, and we can get the adequate system log. The video server cluster stores all the videos and serves five residential districts, all the modification operations on the videos should be performed here. In a residential district, there are about ten residential buildings, and each residential building has one local server that stores the popular video blocks. The local servers are used to share the service pressure of the video server cluster and reduce the access time of the video on demand platform for the users in the residential building. All the servers in a residential district are considered as one server group, and we apply the distributed replication algorithms for such a group.

However, a server group doesn't have the capacity to store all the videos, it can only store the popular videos blocks. In the platform, one video is first encapsulated into some blocks by the package tasks, then these blocks are stored. The size of one block is usually 20 MB. Consequently, the blocks of the popular videos should be replicated by the server group. Meanwhile, since people usually watch the beginning of new videos, the blocks at the beginning of the unpopular videos should also be replicated. Thus, the size of all the popular video blocks is not very large and the server group can satisfy the storage requirements of the popular data in practice. Then we can give the values of t_l, t_r, t_s. Because the block access time in the local server is similar, the block access time in the server group is similar, the block access time in the video server cluster is similar, we use the average block access time in the local server, the average block access time in the server group, the

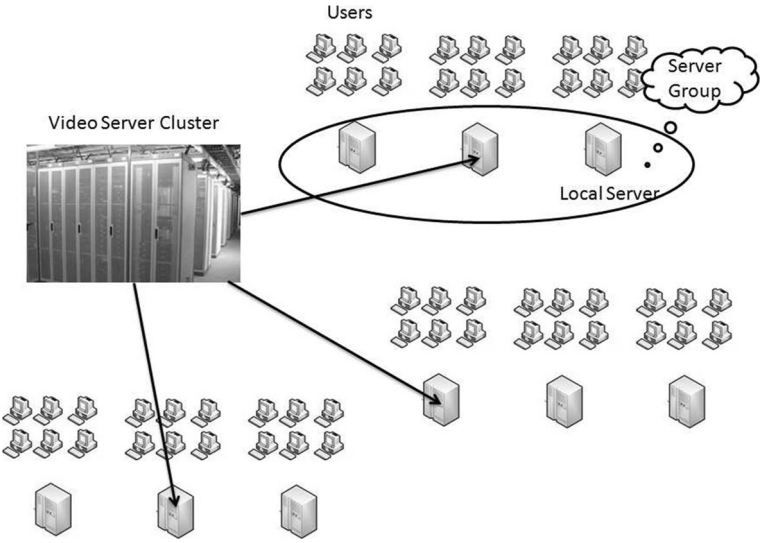

Fig. 2 Experimental environment

average block access time in the video server cluster to denote t_l, t_r, t_s, respectively. In our system, the values of t_l, t_r, t_s are very different from the actual model of [1], and they are calculated as $t_l = 7.19s, t_r = 28.56s, t_s = 373.28s$.

The measurement of the access time is an important problem. The time to get the data from the original server fluctuates dramatically while the time to get the data from the local and group servers fluctuates gently. We consider this problem from two aspects. The most important aspect is that the access time of original server t_s is much larger than the access time of local server t_l and the access time of group server t_r. If this condition doesn't hold, the work of the replication algorithm has no practical meaning. In the experiment, $t_l = 7.19s, t_r = 28.56s, t_s = 373.28s$, even t_s changes dramatically and becomes 273.28 s , it's still much larger than t_l and t_r, this only has a small impact on the overall result. The second aspect is that the average access time of the original server in a period of time fluctuates gently. The average access time can be used to replace the instantaneous access time.

A simulation system is designed to improve the performance of the video on demand platform in our laboratory, the communication environment is set as the actual platform. We select the system log of the past month to simulate the users' behaviors in such a simulation system for the reason that statistical regularities of the users' behaviors will become invalid over time. We extract the demand information from the system log for each popular video block, and the values of request rates $r_{m,n}$ can be counted according to the demand information. The performance gain of the replication algorithms can be given by the following equation according to its definition.

$$PG = \sum_{userID} (t_s - t_{userID})$$

where t_{userID} denotes the actual video block access time of the user with ID *userID*.

The experiments include 2 steps. In the first step, we compare the performances of the FRA and AORA with the optimal algorithm and isolationist algorithm. A fully optimal algorithm is determined by the optimal solution of knapsack problem that has not been solved, so we use a centralized algorithm Aϵ-star proposed in [4] to replace it. The isolationist algorithm work as follows: s_m replicates c_m copies of objects with the highest performance gains independently, the performance gain of o_n is denoted by $r_{m,n}(t_s - t_l)$ for each $1 \le m \le M, 1 \le n \le N$.

In the second step, we compare the performance of the AORA with the Distributed Greedy Replication algorithm(DGR) proposed in [2] and the two-step local search algorithm (TSLS) proposed in [3]. These two algorithms are both synchronous distributed algorithms. The DGR works as follows: In the beginning, all the servers have been fully filled with copies of objects by the initial algorithm. Then in a round, each server calculates the maximum total performance gain of replacement operations of a server, and then the algorithm chooses the maximum total performance gain, and performs the operation. The algorithm continues until the maximum total performance gain is less than 0 in a round. TSLS works as follows: In the beginning, all the servers don't have any copy of objects, and the algorithm sorts all the servers with $1, \ldots, N$. Then in the round m, server m chooses

c_m replication operations with the highest total performance gain for the group, and performs the operations for each $1 \leq m \leq M$. The algorithm stop in the round M. Since the simulation system is a typical distributed replication group and the DGR, TSLS are specially applied to such a system, the fair comparisons can be made in this experimental environment.

Today the trend of video systems is to go towards adaptive streaming protocols that make the server keep several variants of the same video. This technique is called as scalable coding. For example, one scalable video has two copies according to high definition (HD) and standard definition (SD), respectively. A user may request the HD copies of one video if the network condition and the terminal equipment are very good, else the user may request the SD copies. In our model, the AORA is also suitable for this case for the reason that we replicate the copies of the video chunks instead of the video chunks into the servers. One video may have two groups of copies according to HD and SD. The HD and SD copies are very different from each other. The number of SD copies must be far less than that of HD, and one SD copy contains more frames than the HD copy. The weight of one copy is very important in our algorithm. However, the value of the weight is not assigned by our algorithm, the behaviors of the users decide the weight. For example, if the users usually request a copy, this copy is very popular and its weight is large, else the weight is small. For one video, its weights of the SD and HD copies may be different, the SD copy's weight may be large when the HD copy's weight is small. Consequently, the AORA focuses on the copy's weight of one video instead of the video's weight, and it is also suitable for adaptive streaming protocols.

6.2 Experimental Results

In the first step of experiments, we compare the performances of the algorithms under the following situations: 10 servers and 100, 500, and 2,000 objects. We calculate the ratio of the performance gains of the FRA, AORA and isolationist, as well as the optimal algorithms to analysis the performances of the FRA and AORA. In the second step of experiments, we compare the performances, the time and communication overhead of the algorithms under the following situations: 10 servers and 100, 500, and 2,000 objects; 1,000 objects and 5, 10, and 20 servers. The storage capacity of a server is set as 30 % of the number of objects.

Here, we set the number of servers as 10, and adjust the number of objects as 100, 500, and 2,000, respectively. We repeat each comparison experiment 1,000 times. In Table 3, we give the performance comparison results of the FRA and isolationist algorithms. The first column of the table is the number of objects. The second column shows the percentage of comparison results when the performance gain of the AORA is larger than the isolationist algorithm's within the repeated experiments. The third, fourth and fifth columns indicate the minimum, maximum and average of the values of the *FRA/Isolationist*, respectively. The similar descriptions are suitable for Tables 4, 5, 6, 7, 8, 9, 10 and 11.

From the results of Tables 3 and 5, we can see that both the FRA and AORA can obtain better performance gains than the isolationist algorithm. The reason is that both the FRA and AORA consider the total performance gain of the group when

Table 3 Performance comparison of FRA and isolationist

N	FRA > isolationist (%)	Min	Max	Mean
100	100	1.41734	1.45287	1.44010
500	100	1.43629	1.46163	1.44904
2,000	100	1.43261	1.47116	1.45775

Table 4 Performance comparison of FRA and optimal

N	FRA < optimal (%)	Min	Max	Mean
100	100	0.70126	0.96983	0.74054
500	100	0.75024	0.94979	0.77001
2,000	100	0.76216	0.94961	0.78443

Table 5 Performance comparison of AORA and isolationist

N	AORA > isolationist (%)	Min	Max	Mean
100	100	1.77032	1.82221	1.79737
500	100	1.79078	1.83164	1.83143
2,000	100	1.82601	1.85426	1.84517

Table 6 Performance Comparison of AORA and Optimal

N	AORA < optimal (%)	Min	Max	Mean
100	83.1	0.97633	1.00070	0.98117
500	75.9	0.96978	1.00114	0.97925
2,000	81.2	0.97013	1.00197	0.98716

Table 7 Performance comparison of AORA and DGR

M	N	AORA > DGR (%)	Min	Max	Mean
10	100	62.9	0.89982	1.23106	1.06613
10	500	66.2	0.9377	1.31002	1.06972
10	2,000	71.7	0.92978	1.27037	1.07126

Table 8 Performance comparison of AORA and DGR

M	N	AORA > DGR (%)	Min	Max	Mean
5	1,000	73.8	0.87539	1.16026	1.06793
10	1,000	76.7	0.89978	1.24037	1.07431
20	1,000	76.4	0.89997	1.26002	1.07979

Table 9 Performance comparison of AORA and TSLS

M	N	AORA > TSLS (%)	Min	Max	Mean
10	100	100	1.29021	1.44116	1.37334
10	500	100	1.33083	1.43835	1.39127
10	2,000	100	1.32096	1.45983	1.41132

Table 10 Performance Comparison of AORA and TSLS

M	N	AORA > TSLS (%)	Min	Max	Mean
5	1,000	100	1.27827	1.41461	1.35037
10	1,000	100	1.30196	1.44983	1.36832
20	1,000	100	1.33103	1.44647	1.38593

Table 11 Time and communication overhead

M	N	AORA		AORA with FRA		DGR		TSLS	
		Time	Com	Time	Com	Time	Com	Time	Com
10	3,000	15	78	9	41	147	587	24	92
10	6,000	31	124	17	83	319	1,174	37	131
10	9,000	42	196	33	142	642	1,753	44	191
20	3,000	17	136	10	59	294	1,738	34	204
20	6,000	33	268	19	132	682	3,328	67	342
20	9,000	51	412	41	309	1,842	6892	93	677
30	3,000	19	164	9	71	404	3,117	52	493
30	6,000	41	372	22	211	1,227	5,046	88	737
30	9,000	68	693	42	457	2,793	10,132	119	1,039

they make decisions. But in fact, when the size of the group is very small, the first step of the FRA may let the performance be worse than the isolationist in some rare cases. From the results of Tables 4 and 6, we can see the comparison results of the FRA, AORA and the optimal algorithm. If there is a real optimal algorithm, the percentages should be all 100 %, but here we replace it with an approximation algorithm. We can see that the performance of the FRA is worse than AORA, and the performance of the AORA is similar with the optimal algorithm's.

In the second step of experiments, we will compare the AORA with two other distributed algorithms: the TSLS and DGR. We first set the number of servers as ten, adjust the number of objects as 100, 500, and 2,000, respectively, and then set the number of objects as 1,000, adjust the number of servers with 5, 10, and 20, respectively. We repeat each comparison experiment 1,000 times. From Tables 7, 8, 9 and 10, we can see that the performance of the AORA is higher than the TSLS' and DGR's. The DGR is an approximation algorithm, and increases the time complexity to get a high performance, but the high latency also reduces its

performance in our experiments. The TSLS is a distributed selfish algorithm, it can be completed quickly, so the comparison results of the performance is not unexpected. We also plot the frequency distribution for the relative gain of AORA over DGR in Figs. 3 and 4. A bar in these figures represents the percentage of the performance ratio in the specific limit. Figures 3 and 4 show the distributions summarized in Tables 7 and 8. From Fig. 3, we can see that the performance ratio of the AORA and DGR increases as the number of data objects increases. From Figure 4, we can see that the performance ratio of the AORA and DGR increases as the number of servers increases. Meanwhile, we can also observe this slight increase of performance ratio in Tables 7, 8, 9 and 10. By analyzing the performance records of the algorithms, we find that the performance gains of the AORA, DGR, and TSLS for one server or one data object decreases for different data objects or servers, however, the AORA's decreases more slowly.

Then, we compare the time and communication overheads of the AORA, GDR, and TSLS. In Table 11, we can see the comparison results of the time and communication overhead. The first column indicates the number of servers, and the second column indicates the number of objects. The capacity of a server keeps 30 % of the value of N The column with the label Time indicates the time overhead of the algorithm, and the unit of a value is a millisecond. The column with the label Com indicates the communication overhead of the algorithm, and the unit of a value is Kb. From the comparison results, we can see that the time overhead of the AORA

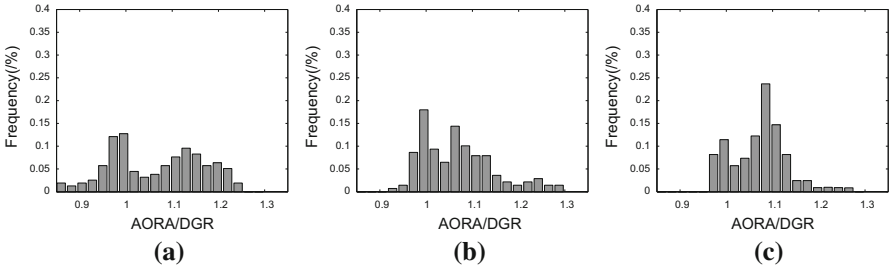

Fig. 3 Distribution of relative gain of AORA over DGR for $M = 10$. **a** $N = 100$, **b** $N = 500$, **c** $N = 2{,}000$

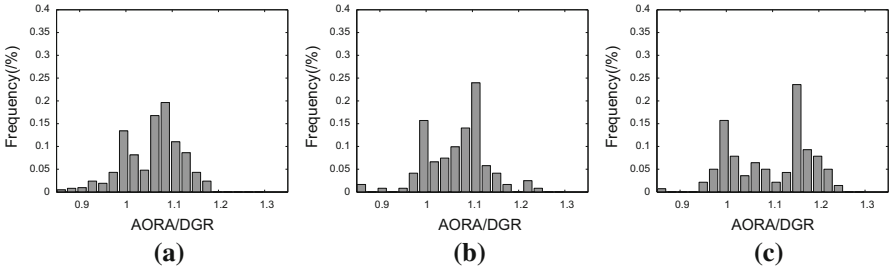

Fig. 4 Distribution of relative gain of AORA over DGR for $N = 1{,}000$. **a** $M = 5$, **b** $M = 10$, **c** $M = 20$

almost doesn't increase with the growth of the number of servers; the reason is that the time complexity of the AORA is not affected by the number of servers M. Meanwhile, since the time complexities of the DGR and TSLS are polynomial with respect to M, the time overhead of the DGR and TSLS have large growth with different rates; the DGR's increases faster, and the TSLS' increases more slowly. Then we consider the experiment results over different values of N, because the maximum capacity of a server c_{max} is indicated by the value of N and the time complexities of the AORA and TSLS are polynomial with respect to c_{max}, the time complexity of the DGR is polynomial with respect to the sum of the capacities of all servers Mc_{max}, all the time overhead of the DGR, TSLS and AORA have large growth. The DGR's increases faster, TSLS' and AORA's increase more slowly. Since the communication complexity is affected by the similar variables that affect the time complexity, the similar comparison results in communication overhead. From Table 10, we can find that the AORA is much faster and requires a less amount of data for the communication than the DGR in all cases, and performs better than the TSLS in most cases. And another important observation in Table 10 is that the AORA with the initial algorithm FRA has greatly reduced both the time and communication overhead for the reason that the FRA can make the storage configuration of the group close to optimal state and guarantee the requirements of the AORA. So the AORA is highly recommended to use the FRA as its initial algorithm.

In the above experiments, the storage capacity of a server is fixed as 30 % of the number of objects for the reason that the number of popular data objects is not very large and the server group can usually meet the storage requirement. Here, we want to explore the case where the server group cannot meet the storage requirement. The storage capacity of one server is set as $c_m = 50$.

Figure 5 shows the frequency distribution for the relative gain of the AORA over the DGR when $c_m = 50$ and $N = 1{,}000$. We can see that the distribution is similar with the distribution of Fig. 4, the performance gain of the AORA is still better. However, we also compare the time and communication overhead in Table 12. Since Theorems 1 and 2 can't be met, the time and communication complexities of the AORA have a great increase; for example, the theoretical time overhead for $M = 30$ and $N = 9{,}000$ should be 1.1s if Theorems 1 and 2 can be satisfied, however, it is now twenty times.

At last, we want to explore the performance gain of the proposed algorithms when the capacities of servers are heterogeneous. We adopt ten servers, and three servers have the storage capacity of 10 % data objects, three servers have the storage capacity of 20 % data objects, four servers have the storage capacity of 30 % data objects. The number of data objects varies over 100, 500, and 2,000. Figure 6 shows the frequency distribution for the relative gain of the AORA over the DGR when the capacity of servers is heterogeneous. We can see that the AORA performs a little worse when the results are compared with Fig. 3. This is because the performance gain of the AORA greatly decreases when its complexity doesn't change according to Theorems 1 and 2.

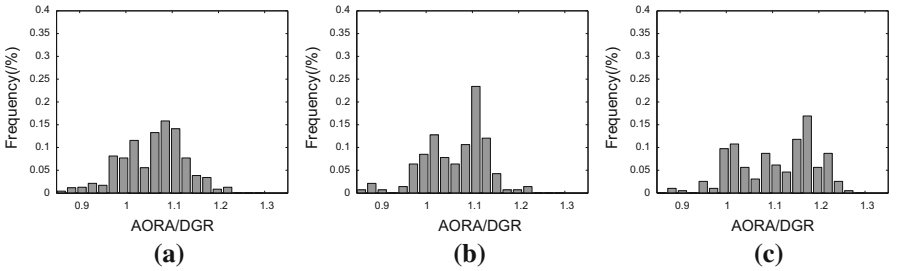

Fig. 5 Distribution of relative gain of AORA over DGR for $c_m = 50$ and $N = 1{,}000$. **a** $M = 5$, **b** $M = 10$, **c** $M = 20$

Table 12 Time and communication overhead for $c_m = 50$

M	N	AORA		DGR	
		Time	Com	Time	Com
10	3,000	1.2	9.7	7.2	31.4
10	6,000	1.7	12.4	7.9	37.9
10	9,000	3.1	21.9	10.3	42.6
20	3,000	2.7	16.2	21.9	183.6
20	6,000	4.8	23.1	31.7	212.2
20	9,000	9.1	41.6	58.4	263.4
30	3,000	8.7	37.2	67.2	459.1
30	6,000	13.5	57.5	98.4	607.6
30	9,000	22.9	113.4	167.6	718.5

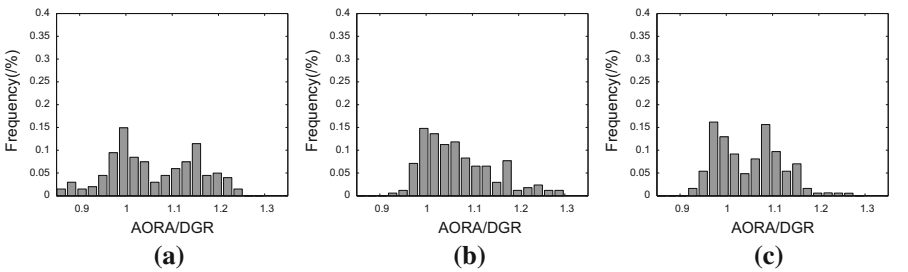

Fig. 6 Distribution of relative gain of AORA over DGR for heterogeneous capacities. **a** $N = 100$, **b** $N = 500$, **c** $N = 2{,}000$

7 Conclusion

An effective replication algorithm can help reduce the access time and improve the performance of services. This work has described two distributed replication algorithms: the FRA and AORA. The FRA is the initial algorithm of the AORA, and can greatly meet its requirements. The AORA is an asynchronous distributed

algorithm with shared memories; it spends more time to complete the configuration, but its result will approach the optimal algorithm's. We give detailed proofs of the complexity of the AORA, and the proofs show us that the time complexity has been reduced by the times of the number of servers when the performance is similar with the DGR's. The results of the experiments also show us that the FRA can be completed in a short period of time and reach 77 % performance of the optimal algorithm, and the AORA can be completed much faster than the DGR, and obtain 98 % performance of the optimal algorithm.

However, there are still some problems that can be considered to improve the distributed replication algorithms. First, from the simulation results, we can find that the time and communication complexities of AORA significantly increase if the storage capacity of the group can't meet the requirements. Second, the replication algorithms need the values of t_l, t_r, t_s to allocate the data objects. If the data sizes of the objects are very different from each other, the values of t_l, t_r, t_s can't be calculated, and the package technique may be used to solve this problem. Third, an adequate system log is required for the reason that the statistic regularities of access rates are needed to complete the replication algorithm.

Acknowledgments This work was supported by Key Program of National Natural Science Foundation of P.R. China under Grant No.61233003 and China Postdoctoral Science Foundation under Grant No. 2014M561839.

References

1. Leff, A., Wolf, J.L., Yu, P.S.: Replication algorithms in a remote caching architecture. IEEE Trans. Parallel Distrib. Syst. **4**(11), 1185–1204 (1993)
2. Zaman, S., Grosu, D.: A distributed algorithm for the replica placement problem. IEEE Trans. Parallel Distrib. Syst. **22**(9), 1455–1468 (2011)
3. Laoutaris, N., Telelis, O., Zissimopoulos, V., Stavrakakis, I.: Distributed selfish replication. IEEE Trans. Parallel Distrib. Syst. **17**(12), 1401–1413 (2006)
4. Khan, S.U., Ahmad, I.: Comparison and analysis of ten static heuristics-based Internet data replication techniques. J Parallel Distrib. Comput. **68**(2), 113–136 (2008)
5. He, D., Liang, Y., Hang, Z.: Replicate distribution method of minimum cost in cloud storage for Internet of things. In: Proceedings IEEE symposium network computing and information security (NCIS), pp. 89–92 (2011)
6. Krishnan, P., Raz, D., Shavitt, Y.: The cache location problem. IEEE/ACM Trans. Netw. **8**(5), 568–582 (2000)
7. Tang, B., Gupta, H., Das, S.R.: Benefit-based data caching in ad hoc networks. IEEE Trans. Mob. Comput. **7**(3), 289–304 (2008)
8. Kangasharju, J., Roberts, J., Ross, K.W.: Object replication strategies in content distribution networks. Comput. Commun. **25**(4), 376–383 (2002)
9. Baev, I., Rajaraman, R., Swamy, C.: Approximation algorithms for data placement problems. SIAM J. Comput. **38**(4), 1411–1429 (2008)
10. Li, B., Golin, M.J., Italiano, G.F., Deng, X., Sohraby, K.: On the optimal placement of web proxies in the Internet. In: Proceedings of conference computer communication (IEEE INFOCOM) (1999)
11. Changjie, G., Zhe, X., Yuzhuo, Z.: A novel greedy heuristic placement algorithm in distributed cooperative proxy systems. Proc. IEEE Symp. Info-Tech Info-Net **2001**(5), 229–234 (2001)
12. Kalpakis, K., Dasgupta, K., Wolfson, O.: Optimal placement of replicas in trees with read, write, and storage costs. IEEE Trans. Parallel Distrib. Syst. **12**(6), 628–637 (2001)
13. Fadelelmoula, A.A., Dominic, P.D.D.: Optimistic replication in mobile traffic control environment. Intelligent and Advanced Systems (ICIAS 2007), pp. 543–548 (2007)

14. Keqiu, L., Hong, S., Chin, F.Y.L., Weishi, Z.: Multimedia object placement for transparent data replication. IEEE Trans. Parallel Distrib. Syst. **18**(2), 212–224 (2007)
15. Zhou, J., Wang, Y., Li, S.: An optimistic replication algorithm to improve consistency for massive data. Lecture Notes in Computer Science, pp. 713–718 (2005)
16. Loukopoulos, T., Ahmad, I.: Static and adaptive data replication algorithms for fast information access in large distributed systems. Proc. IEEE Symp. Distrib. Comput. Syst. **2000**, 385–392 (2000)
17. Loukopoulos, T., Ahmad, I.: Static and adaptive distributed data replication using genetic algorithms. J. Parallel Distrib. Comput. **64**(11), 1270–1285 (2004)
18. Xin, S., Jun, Z., Qiongxin, L., Yushu, L.: Dynamic data replication based on access cost in distributed systems. In: Proceedings of IEEE symposium computer sciences and convergence information technology (ICCIT '09), pp. 829–834 (2009)
19. Triantafillou, P., Taylor, D.J.: Multiclass replicated data management: exploiting replication to improve efficiency. IEEE Trans. Parallel Distrib. Syst. **5**(2), 121–138 (1994)
20. Yi-Hsuan, F., Nen-Fu, H., Yen-Min, W.: Efficient and adaptive stateful replication for stream processing engines in high-availability cluster. IEEE Trans. Parallel Distrib. Syst. **22**(11), 1788–1796 (2011)
21. Zhang, W.: Replication cache: a small fully associative cache to improve data cache reliability. IEEE Trans. Comput. **54**(12), 1547–1555 (2005)
22. Xueyan, T., Chanson, S.T.: Minimal cost replication of dynamic Web contents under flat update delivery. IEEE Trans. Parallel Distrib. Syst. **15**(5), 431–439 (2004)
23. Shen, K., Yang, T., Chu, L.: Clustering support and replication management for scalable network services. IEEE Trans. Parallel Distrib. Syst. **14**(11), 1168–1179 (2003)
24. Shi, Z., Beard, C., Mitchell, K.: Analytical models for understanding misbehavior and MAC friendliness in CSMA networks. Perform. Eval. **66**(9–10), 469–487 (2009)
25. Shi, Z., Beard, C., Mitchell, K.: Analytical models for understanding space, backoff and flow correlation in CSMA wireless networks. Wirel. Netw. **19**, 393–409 (2013)
26. Shi, Z., Beard, C., Mitchell, K.: Competition, cooperation, and optimization in multi-hop CSMA networks with correlated traffic. Int. J. Next-Gener. Comput. **3**(3), 117–120 (2012)
27. Shi, Z.: Stochastic modeling, correlation, competition, and cooperation in a CSMA wireless network. ProQuest, UMI Dissertation Publishing (2011)
28. Shi, Z., Gu, R.: Efficient implementation of particle swarm optimization algorithm. Int. J. Soft Comput. Math. Control. **2**(4), 1–13 (2013)
29. Shi, Z., Gu, R.: A framework for mobile cloud computing selective service system. IEEE 2013 Wireless Telecommunications Symposium, pp. 1–5 (2013)
30. Shi, Z., Beard, C.: QoS in the mobile cloud computing environment, mobile computing over cloud: technologies, services, and applications (2013)

Xiaofeng Jiang received the B.E. and Ph.D. in information science and technology from University of Science and Technology of China (USTC), Hefei, China, in 2008 and 2013. He is a postdoctoral research associate at the department of automation in information science and technology of USTC, and joined the Laboratory of Network Communication System and Control in 2013. His recent research interests include distributed algorithm, discrete event dynamic system, and wireless communications.

Jun Li received the B.S. and Ph.D. degrees in information science and technology from University of Science and Technology of China (USTC), Hefei, China, in 1996 and 2001, respectively. He is an expert in the field of information science, and has completed 2 state 863 projects: development of streaming media server under Grant No. 2003AA103710 and development of large scale concurrent streaming media server under Grant No. 2005AA103320. His research interests include advanced control theory and applications and network communication system and control.

Hongsheng Xi received the B.S. and M.S. in applied mathematics from University of Science and Technology of China (USTC), Hefei, China, in 1980 and 1985. He is currently the Dean of the School of Information Science and Technology, USTC. He is a professor in department of automation and also directs the Laboratory of Network Communication System and Control. His research interests include stochastic control systems, discrete event dynamic systems, network performance analysis and optimization, and wireless communications.