Tsinghua Science and Technology
Doi: xxxx

Research Article

Stage-aware Bayesian-guided evolutionary algorithm for resource constrained serverless MapReduce scheduling

Chuanzhi Chen, Hongyan Sang (✉), *Senior Member, IEEE*, Biao Zhang, Jinquan Zhang, and Hengwei Guo

Abstract Serverless computing features instant elasticity and fine-grained billing. It has emerged as a prominent paradigm for big data processing such as log analysis and monitoring. However, scheduling MapReduce jobs under serverless resource constraints is challenging because the stateless nature of functions conflicts with MapReduce stage dependencies, which increases scheduling complexity. Existing research has not adequately captured these structural constraints, causing traditional evolutionary algorithms to explore the search space inefficiently. This study addresses the MapReduce scheduling problem under serverless resource constraints with the objective of minimizing the makespan. Accordingly, a stage-aware Bayesian-guided evolutionary algorithm (BGA) is proposed that, unlike traditional evolutionary algorithms, guides evolution through Bayesian learning from elite patterns. First, a mixed-integer programming model is formulated to unify temporal and precedence constraints. Second, a Bayesian probability model for assigning tasks to function instances guides evolutionary sampling using elite patterns, reducing inefficient search. Finally, dependency-aware crossover is integrated with adaptive variable neighborhood search to ensure feasibility and enhance local exploitation. Experimental results show that BGA consistently achieves the lowest makespan among compared methods, with Relative Percentage Deviation (RPD) remaining within 2.40%–3.92%.

Keywords Serverless MapReduce scheduling; Bayesian method; variable neighborhood search; large-scale workflow optimization

• Chuanzhi Chen, Hongyan Sang, and Biao Zhang are with the School of Artificial Intelligence and Computer Science, Liaocheng University, Liaocheng 252000, China. E-mail: chencz8023@163.com; sanghongyan@lcu-cs.com; zhangbiao@lcu-cs.com.
• Jinquan Zhang is with the School of Computer Science and Technology, Guangdong University of Technology, Guangzhou 510006, China. E-mail: zjq8860@gdut.edu.cn.
• Hengwei Guo is with the School of Mechatronic Engineering and Automation, Shanghai University, Shanghai 200444, China. E-mail: guohengwei@shu.edu.cn.
* Corresponding author: Hongyan Sang.
Manuscript received: 20xx-xx-xx; revised: 20xx-xx-xx; accepted: 20xx-xx-xx

1 Introduction

Serverless Function as a Service (FaaS) is a new resource provisioning paradigm following virtual machines and containers. It provides event-driven execution, instant elasticity, and per-invocation billing [1]. These features enable zero idle cost and rapid scaling for data-intensive workloads, making serverless computing an attractive platform for large-scale data processing. MapReduce workflows for log analysis and monitoring are one important class of such applications and are increasingly deployed on serverless platforms [2]. In these applications, timely completion of jobs is important for maintaining accurate and up-to-date insights, so job completion time is a prominent performance concern for the scheduler. Scheduling such MapReduce workflows on serverless platforms to minimize makespan is therefore a problem worth studying.

MapReduce is a parallel data analytics processing model [3]. A typical job contains multiple parallel map tasks and several reduce tasks. Each map task handles one data partition. Reduce tasks then merge the map outputs to generate the final result, and multiple jobs can be chained into workflows. Map tasks form the first layer, and reduce tasks form the second, defining a strict dependency structure where reduce tasks must wait for all dependent map tasks to complete before executing.

However, the scheduling of serverless MapReduce is challenging. The stateless execution model conflicts with MapReduce stage dependencies: functions cannot retain intermediate data locally, requiring all map outputs to be transferred through external storage before reduce tasks can access them. This precedence-constrained communication pattern creates a tightly constrained search space. The scheduling problem is NP-hard [4]. Traditional evolutionary algorithms struggle with such constrained search spaces.

This work investigates MapReduce task scheduling under serverless resource constraints in scenarios such as log analysis and monitoring. The objective is to minimize the makespan. The problem is to schedule map and reduce tasks on serverless function instances, hereafter called function instances. Specifically, the scheduler allocates map and reduce tasks to function instances under precedence and resource

Just Accepted

constraints, aiming to find near-optimal task assignments and execution orders.

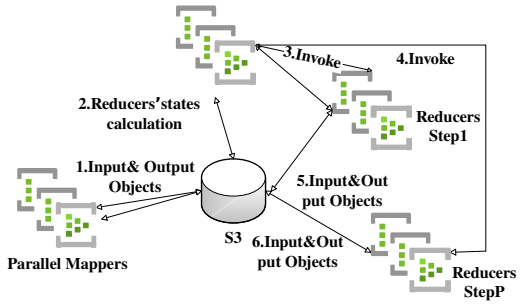


Fig. 1 Serverless MapReduce scheduling framework

Fig. 1 illustrates the scheduling framework. Despite extensive research on workflow scheduling, existing solutions provide limited support for serverless MapReduce, where stage dependencies conflict with stateless execution. Traditional algorithms, such as cost-efficient scheduling with deadline constraint (CETSS) proposed by Tang et al. [5] and the Poset-based method by Nadjafi-Arani et al. [6], primarily target virtual machines or containers. These methods assume controllable states and long-term online resources, which limits their applicability to the ephemeral nature of serverless functions.

Recent work has begun to address serverless environments, existing studies have primarily considered isolated aspects such as I/O optimization [2], cold-start modeling [1], and energy awareness [7]. However, these studies lack unified frameworks that address MapReduce stage dependencies and scale to large problem instances in serverless environments.

In summary, serverless MapReduce workflow scheduling still faces three challenges.

1) Existing models do not adequately capture MapReduce stage dependencies and have objectives that lack structural awareness.

2) The feasible search space is strongly constrained by precedence relationships, causing traditional evolutionary operators to frequently generate infeasible offspring. Moreover, random exploration discards knowledge from elite solutions, which leads to inefficient search.

3) Traditional evolutionary algorithms suffer from premature convergence, which limits their effectiveness to solve large-scale scheduling problems.

To address these challenges, this work proposes the stage-aware Bayesian-guided evolutionary algorithm (BGA). First, a stage-aware mixed-integer programming model is formulated that unifies MapReduce precedence, stateless execution, and resource constraints, providing a unified basis for feasibility analysis.

Second, to overcome the inefficiency of random exploration, a Bayesian guidance mechanism is introduced that learns frequent patterns of assigning tasks to function instances from elite solutions and combines them with performance priors over function instances to guide offspring sampling.

Third, to improve feasibility, a dependency-aware crossover operator is developed together with a three-level variable neighborhood search strategy. This enables multi-scale exploration over assignment and ordering decisions within the feasible space.

2 Related work

2.1 MapReduce scheduling in traditional environments

MapReduce is widely used for distributed data processing [8]. Researchers have proposed various scheduling methods for MapReduce, including graph-based approaches such as exact and approximate models [4] and pairing or overlapping strategies for multi-job scenarios [9]. These methods effectively reduce makespan and improve data locality, but become computationally expensive for large-scale problems.

As MapReduce problem scales increase, exact methods become difficult to solve within reasonable time. Metaheuristic algorithms have proven effective for various scheduling problems [10–17]. These methods have been widely used in MapReduce scheduling. Jeyaraj and Paul [18] applied ant colony optimization (ACO) to improve data locality and load balance. Demirbaga [19] proposed EcoCloud, an ACO-based mechanism enhanced with multilayer perceptron neural networks for energy-efficient MapReduce task scheduling in Hadoop clusters. Diwaker et al. [20] proposed MapReduce-Modified Particle Swarm Optimization (MR-MPSO) to improve scheduling efficiency by the adaptive inertia weight adjustment. Additionally, Bergui et al. [21] applied machine learning techniques to predict MapReduce job performance for scheduling optimization.

Overall, traditional research comprises exact and approximate optimization methods, metaheuristic algorithms, and performance prediction techniques. These methods generally assume stable resources and long-lived instances, where node lifetime, startup delay, and resource location are controllable or predictable [4, 5]. When underlying resources shift to short-lived, on-demand instances, as in serverless platforms, the validity of these assumptions and methods declines.

Just Accepted

2.2 Scheduling problems in serverless environments

Compared with traditional platforms, serverless functions are created and released on demand. Their execution performance is affected by startup overhead, resource allocation, and workload dynamics [1]. Existing scheduling studies in serverless environments can be broadly grouped into function-level resource management and workflow-level orchestration.

Function-level resource management focuses on improving the runtime conditions of individual function invocations. Mampage et al. proposed a learning-based scaling method to optimize execution time and resource cost [22]. Chen et al. developed a Cache-Based Adaptive Scheduler for Serverless Runtime to mitigate cold starts and improve the utilisation of resources [23]. Other works address request dispatching [24], energy-aware resource allocation [25], and performance–cost optimization [26]. These studies capture important runtime factors, but they typically optimize individual function invocations without modeling inter-task dependencies.

Workflow orchestration considers dependencies among functions. Feng et al. studied the resource configuration and proactive elastic allocation for serverless workflow applications [27]. Zhang et al. jointly determined the task execution order, resource leasing, and mapping of task for budget-constrained workflows by using cloud servers and serverless resources [28]. Cheng et al. studied SLO-aware function placement for serverless workflows with static and dynamic DAG structures [29]. Zolfaghari et al. combined evolutionary algorithms with heuristics for dynamic function placement and request scheduling in serverless workflows [30]. These studies consider task dependencies, yet they address general workflows and do not provide specialized scheduling strategies for computations with fixed stage structures and bulk inter-stage data transfers.

2.3 The MapReduce scheduling problem in serverless environments

When MapReduce is migrated to serverless platforms, stage dependencies, stateless execution, and resource heterogeneity create new scheduling challenges. Stage dependencies conflict with stateless execution. Because reduce tasks depend on intermediate outputs from the map phase, these data must be transferred through external storage, which increases I/O overhead and long-tail latency [2]. Additionally, resource heterogeneity introduces scheduling challenges. Function instances exhibit varying processing rates, affecting task execution times and making optimal task-to-instance assignment more complex.

These challenges have motivated specialized research on serverless MapReduce and data processing. Huang et al. [2] proposed an optimization framework, which reduces execution time by 25.6% and cost by 17.3% by mitigating inefficient scheduling, high shuffle-read overhead, and long-tail I/O latency. Jarachanthan et al. proposed Astrea, a graph-theoretic orchestration framework for FaaS-based MapReduce that reduces execution time by 21%–69% and cost by 20%–84% on AWS Lambda [31]. A follow-up study further improved its A* algorithm and achieved roughly 30% higher solving efficiency for serverless MapReduce resource allocation [32]. Zhou et al. [33] proposed StarData, a lightweight MapReduce framework built on AWS Lambda that optimizes parallel processing through adaptive adjustment of function count, file size, and resource allocation.

While these serverless MapReduce methods address aspects such as cost optimization or resource allocation, they do not jointly optimize task assignment considering both stage dependencies and resource heterogeneity. To address this gap, this work proposes BGA for serverless MapReduce scheduling.

3 Serverless environment and MapReduce scheduling model

3.1 Problem description

Consider a serverless MapReduce workflow for large-scale data analytics, such as log analysis and monitoring [2, 31]. These applications are highly sensitive to completion time; this paper therefore aims to minimize the workflow makespan. Raw data are partitioned into multiple blocks; each block is processed by a map task to produce intermediate key–value pairs, which are then aggregated by reduce tasks to produce final statistics.

MapReduce comprises map and reduce as two computational stages. The effect of shuffle phase is captured in dependency constraints and data transfer overhead reflected in processing rates.

In the map phase, map tasks can be executed in parallel on multiple function instances, but parallelism is bounded by account quotas and instance limits. In the reduce phase, each reduce task can start only after its dependent map tasks finish, which forms the precedence constraints of the workflow. Because each task runs as a function invocation on dynamically allocated instances, the scheduler must determine task assignment and execution order under these resource and precedence constraints.

The model needs to abide by the following three constraints.

Just Accepted

1) Each task, whether map or reduce, must be assigned to exactly one function instance.

2) Each function instance can process at most one task at a time, and once a task starts, preemption is not permitted.

3) Task failure is not considered [34].

The serverless MapReduce scheduling problem is formulated as a resource and dependency constrained scheduling problem. Consider a workflow task set $T = M \cup R$, comprising a map set $M = \{m_1, \ldots, m_{|M|}\}$ and a reduce set $R = \{r_1, \ldots, r_{|R|}\}$. These tasks are executed on a set of n function instances $N = \{N_1, \ldots, N_n\}$ where each function instance N_j operates at a processing rate s_j. The objective is to determine an optimal assignment X that minimizes the workflow makespan.

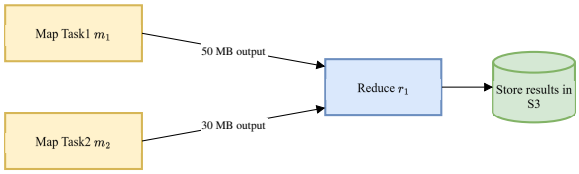


Fig. 2 MapReduce phase workflow

3.2 Problem formulation

Fig. 2 illustrates the MapReduce phase workflow with map and reduce stages. The parameters are defined in Table 1. The set of function instances is denoted by $N = \{N_1, \ldots, N_n\}$, each with a processing rate s_j (MB/ms). The task set $T = M \cup R$, each map task $m_i \in M$ has input data size d_{m_i}. The input of each reduce task $r_k \in R$ is computed from the map outputs with added random perturbation. The dependency set from reduce to map is denoted by $D \subseteq R \times M$. If $(r, m) \in D$, this indicates that reduce task r depends on map task m. The dependency set is given by the workflow instance and is used to verify scheduling feasibility. Note that the model is general and applies to tasks with arbitrary data sizes, as demonstrated in the experimental section.

3.2.1 *Map phase*

Queueing for startup is

$$q_{m_i}^{\text{start}} = F_{N_j}^{\text{cpu}} \tag{1}$$

where $F_{N_j}^{\text{cpu}}$ denotes the last release time of the function instance's CPU slot.

If task T_i is assigned to function instance N_j, its execution time $\tau_{m_i}^{\text{exe}}$ is

$$\tau_{m_i}^{\text{exe}} = \frac{d_{m_i}}{s_j} \tag{2}$$

Table 1 Parameter Definitions

Notation	Description
N	Set of function instances
N_j	A specific function instance
n	Number of function instances ($\|N\| = n$)
s_j	Processing rate of function instance N_j (MB/ms)
M, R	Map/Reduce task set
d_i	Task i input data size
T	Task set
m	Number of tasks ($\|T\| = m$)
$x_{i,n}$	Assignment binary variable (task i, instance n)
τ_i^{exe}	Execution time of task i
q_i^{start}	Task i queue start time
C_i	Task i completion time
$F_{N_j}^{\text{cpu}}$	Last release time of CPU slot
$\pi_{i,j,n}$	Ordering binary variable
$C_{\max}$	Global completion time
P_s	Population size
p_m	Mutation probability
$T_{\max}$	Clock limit
P_t, Q_t	Parent/offspring population at generation t
g_{best}	Global best individual
$P_{i \to N_j}$	Task–instance Bayesian probability

The map output is immediately placed into the shuffle buffer, and its completion time is recorded.

$$C_{m_i} = q_{m_i}^{\text{start}} + \tau_{m_i}^{\text{exe}} \tag{3}$$

3.2.2 *Reduce phase*

Task start time. Reduce task r_k can start only after all dependent map outputs have arrived.

$$q_{r_k}^{\text{start}} = \max\left\{C_{m_i}, F_{N_j}^{\text{cpu}}\right\} \tag{4}$$

If task r_k is assigned to function instance N_j, its execution time is

$$\tau_{r_k}^{\text{exe}} = \frac{d_{r_k}}{s_j} \tag{5}$$

Completion time is

$$C_{r_k} = q_{r_k}^{\text{start}} + \tau_{r_k}^{\text{exe}} \tag{6}$$

Update release time is

$$F_{N_j}^{\text{cpu}} \leftarrow C_{r_k} \tag{7}$$

3.2.3 *Decision variables and constraints*

Task assignment constraint.

Each task must be assigned to exactly one function instance. The binary variable $x_{i,n} = 1$ if task i is assigned to instance n, otherwise $x_{i,n} = 0$.

$$\sum_{n \in N} x_{i,n} = 1, \quad \forall i \in T \tag{8}$$

The task execution time τ_i^{exe} includes data transfer time.

$$\tau_i^{\text{exe}} = \sum_{n \in N} \left(\frac{d_i}{s_n} \cdot x_{i,n}\right), \quad \forall i \in T \tag{9}$$

Just Accepted

In this formulation, d_i/s_n denotes the effective execution time, where platform-level factors such as cold start delays and I/O overhead are abstracted into the processing rate s_n. Under serverless pay-per-use billing, minimizing makespan with fixed task assignments reduces overall execution cost.

The completion time of a task is defined as the sum of its start time and execution duration. The function instance's CPU release time is updated to this completion time.

$$C_i = q_i^{\text{start}} + \tau_i^{\text{exe}}, \quad \forall i \in T \tag{10}$$

Dependency constraint. A reduce task r_k can start only after the outputs of all its dependent map tasks have arrived.

$$q_{r_k}^{\text{start}} \geqslant C_{m_i}, \quad \forall (r,m) \in D \tag{11}$$

Non-overlap constraint on each function instance.

Tasks assigned to the same function instance must follow a sequential order, $\pi_{i,j,n} \in \{0,1\}$. If tasks i and j are both assigned to function instance n, then $\pi_{i,j,n} = 1$ indicates that task i is executed before task j.

$$\pi_{i,j,n} + \pi_{j,i,n} \geqslant x_{i,n} + x_{j,n} - 1, \quad \forall n \in N, \forall i \neq j \in T \tag{12}$$

Linearized ordering constraints use a Big-M [35] constant M_{big}, which is chosen sufficiently large to linearize the time-ordering logic.

$$C_i \leqslant q_j^{\text{start}} + M_{big} \cdot (1 - \pi_{i,j,n}), \quad \forall n \in N, \forall i \neq j \in T \tag{13}$$

$$C_j \leqslant q_i^{\text{start}} + M_{big} \cdot (1 - \pi_{j,i,n}), \quad \forall n \in N, \forall i \neq j \in T \tag{14}$$

Variable domains.

$$x_{i,n}, \pi_{i,j,n} \in \{0,1\} \tag{15}$$

In summary, let $T = \{T_1, \ldots, T_m\}$ be the task set and $N = \{N_1, \ldots, N_n\}$ be the set of function instances. For each task T_i, the data size is d_i, the eligible set of function instances is $E_i \subseteq N$, and the dependency set is $D_i \subseteq T$. For each function instance N_j, the processing rate is s_j (MB/ms). Assign each task T_i to a function instance N_j to form a schedule $\theta = \{a_1, a_2, \ldots, a_m\}$.

3.3 Objective function

The makespan is defined as

$$C_{\max}(\theta) = \max_{j=1}^{|N|} F_{N_j}^{\text{cpu}} \tag{16}$$

Therefore, the scheduling problem is formulated as an optimization problem

$$f(x) = \min_{\theta} C_{\max}(\theta) \tag{17}$$

The model is a multiprocessor scheduling problem with precedence constraints that is NP-hard, where the complexity mainly arises from the task ordering variables $\pi_{i,j,n}$. Verification with CPLEX on small instances confirms the model correctness, while also revealing that the computation cost increases rapidly with task scale.

4 Bayesian-guided evolutionary scheduling algorithm

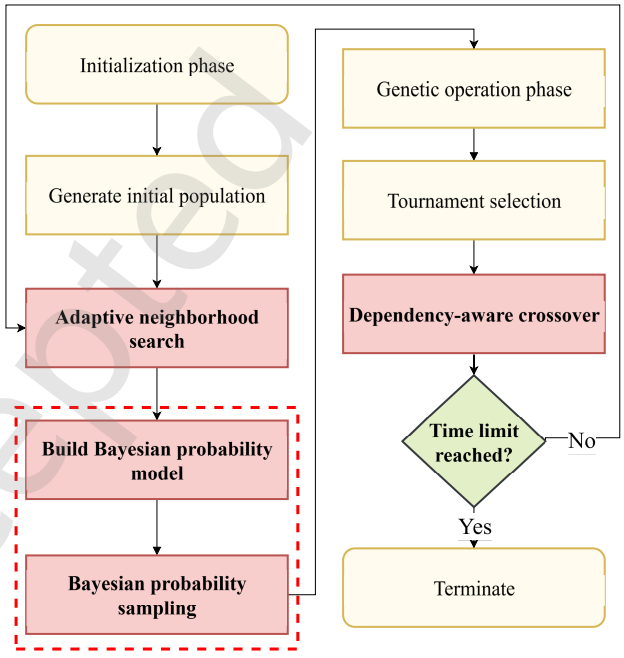


Fig. 3 BGA scheduling framework

This section presents BGA for serverless MapReduce workflow scheduling. This paper takes the genetic algorithm as an illustrative example. It first outlines the overall procedure, then describes key components including the encoding scheme, fitness definition, Bayesian network construction and sampling, and next generation production.

4.1 Framework of BGA

Standard genetic algorithms (GA) suffer from inefficient search and premature convergence in serverless MapReduce scheduling. To address these challenges, the proposed BGA framework has three innovations. First, a Bayesian probability model learns from elite solutions and guides offspring generation. Second, dependency-aware crossover enforces topological integrity during recombination to prevent infeasible solutions caused by precedence violations. Third, adaptive variable neighborhood search (VNS) performs multi-scale local refinement to avoid premature convergence. Algorithm 1 formalizes this process, and Fig. 3 illustrates the overall framework.

Just Accepted

Algorithm 1 BGA Algorithm

Input: population size P_s; mutation rate P_m; $T_{\max}$
Output: g_{best}, Makespan

$P_0 \leftarrow$ Initialization(P_s);
$g_{\text{best}} \leftarrow \arg\min_{e \in P_0}$ Eval(e);
while clock$() - t_{\text{start}} < T_{\max}$ **do**
  if gen $\bmod I_{\text{vns}} = 0$ **then**
    foreach $e \in P[0 : 0.1 \times P_s]$ **do**
      $e \leftarrow$ VNS_Optimize$(e, k_{\max})$;
    end foreach
  end if
  if gen $\bmod 10 = 0$ **then**
    $B \leftarrow$ build Bayes Model from elite solutions;
  end if
  $P' \leftarrow$ $\{g_{\text{best}}\} \cup$ generate Bayes Offspring$(B, R_b \times P_s)$;
  while $|P'| < P_s$ **do**
    $c \leftarrow$ DependencyAwareCrossover(tournament Select(P), tournament Select(P));
    mutate(c, p_m); $P' \leftarrow P' \cup \{c\}$;
  end while
  $P \leftarrow P'$; $g_{\text{best}} \leftarrow \arg\min_{e \in P}$ Eval(e);
  gen $\leftarrow$ gen $+ 1$;
end while
return g_{best}, Makespan;

4.2 Encoding and decoding

The schedule is encoded as a mapping from tasks to function instances. Let $T = \{T_1, \ldots, T_m\}$ denote the task set and $N = \{N_1, \ldots, N_n\}$ the function instance set. Each individual's chromosome is a vector $\theta = \{T_1 \rightarrow N_{j_1}, T_2 \rightarrow N_{j_2}, \ldots, T_m \rightarrow N_{j_m}\}$, where $N_{j_i} \in E_i$ and E_i is the feasible instance set for task T_i. Fig. 4 shows the encoding scheme with three layers. The top layer shows the dependency relations. Task r_1 depends on m_1 and m_3. Task r_3 depends on m_3, m_4, and m_5. Task r_2 depends on m_2. The middle layer presents the task sequence arranged in topological order. The bottom layer represents the task-to-instance assignment.

The makespan is computed by decoding θ into a schedule. The decoding procedure follows the topological order shown in the task sequence layer to respect task dependencies. For each task in the sequence, its start time is determined as the larger value between the completion times of all predecessor tasks and the available time of the assigned function instance. The execution time is computed by dividing the task's data size by the processing rate of the assigned instance. This decoding process ensures that all precedence constraints are satisfied while computing the makespan.

To balance exploration and exploitation, the initial population uses a hybrid strategy: 30% of tasks are assigned greedily to favor faster instances for larger tasks, while 70% are randomly assigned.

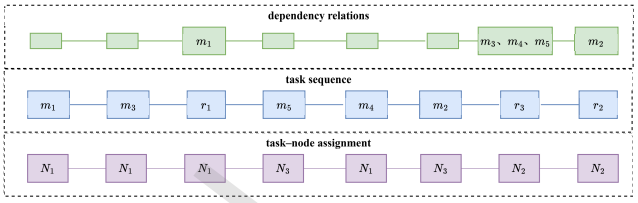


Fig. 4 Encoding scheme

4.3 Bayesian probability model

The feasible space increases exponentially with the number of tasks, rendering exhaustive search intractable. To address this challenge, a Bayesian probability model is integrated to guide the search. This model learns high-frequency patterns from elite solutions, incorporates performance priors to favor faster function instances, and maintains exploration diversity through probabilistic selection.

Let $P(N_j \mid T_i)$ denote the probability that task T_i should be scheduled to function instance N_j. New individuals are generated by sampling from this model.

Step 1. Build a frequency-based model for assigning tasks to function instances.

At generation g, the algorithm keeps the top α elite individuals and denotes them as set E. Let $f_{ij}^{(g)}$ be the number of times task T_i is assigned to function instance N_j. Laplace smoothing is applied to obtain the empirical frequency.

$$\acute{f}_{ij}^{(g)} = f_{ij}^{(g)} + \epsilon \tag{18}$$

where $\epsilon > 0$ is a smoothing factor that prevents zero probabilities. This value reflects whether a task tends to be scheduled to a given function instance.

Step 2. Introduce performance over function instances as a prior.

If two function instances have the same empirical frequency, the faster function instance should be preferred. A performance weight is defined for each function instance as

$$\beta_j = \frac{s_j}{\bar{s}} \tag{19}$$

$$\bar{s} = \frac{1}{|N|} \sum_{k \in N} s_k \tag{20}$$

where $\beta_j > 1$ means that function instance j performs better than the average.

Step 3. Combine the posterior probability.

Just Accepted

The posterior probability is proportional to the likelihood times the prior,

$$P(N_j \mid T_i) \propto P(T_i \mid N_j) \cdot P(N_j) \tag{21}$$

Frequency preference and performance prior are combined to derive the final scheduling probability. Here, $P(N_j) \propto \beta_j$. The probability that task T_i is assigned to function instance N_j is defined as

$$P(N_j \mid T_i) = \frac{\acute{f}_{ij}^{(g)} \cdot \beta_j}{\sum_{k \in N} \acute{f}_{ik}^{(g)} \cdot \beta_k} \tag{22}$$

The formulation combines empirical frequency and performance prior, normalized to ensure probabilities sum to one.

Step 4. Sampling strategy for generating new solutions.

When constructing offspring in the next generation, a function instance is sampled for each task T_i from the posterior distribution $P(N_j \mid T_i)$ as

$$a_i \sim \text{Categorical}(P(N_1 \mid T_i), \ldots, P(N_{|N|} \mid T_i)) \tag{23}$$

This yields a scheduling plan

$$\theta^{\text{new}} = \{a_1, a_2, \ldots, a_m\} \tag{24}$$

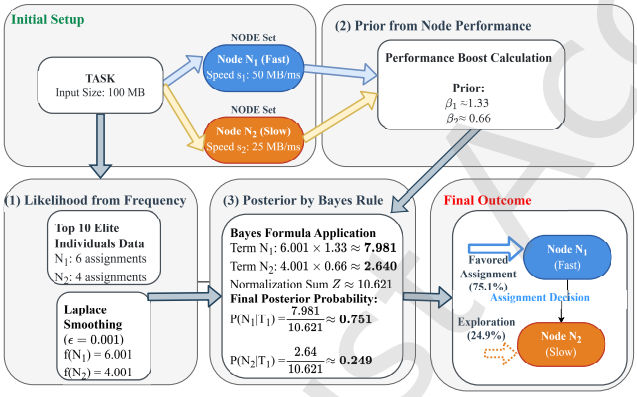


Fig. 5 Schematic diagram of Bayesian task scheduling

Fig. 5 illustrates a case with task set $T = \{T_1\}$ and function instances $N = \{N_1, N_2\}$. Task T_1 has input size 100 MB. The processing rates are $s_1 = 50$ MB/ms (fast) and $s_2 = 25$ MB/ms (slow).

1) Likelihood from frequency. Among the top 10 elite individuals, task T_1 is assigned as N_1:6 and N_2:4. With smoothing $\epsilon = 0.001$, the empirical likelihood is $\acute{f}_{11} = 6 + 0.001 = 6.001$, $\acute{f}_{12} = 4 + 0.001 = 4.001$.

2) Prior from performance over function instances. A performance boost is computed for each function instance. $\bar{s} = \frac{s_1+s_2}{2} = \frac{50+25}{2} = 37.5$. Thus, $\beta_1 = \frac{50}{37.5} = 1.\overline{3}$, $\beta_2 = \frac{25}{37.5} = 0.\overline{6}$.

3) Posterior by Bayes rule. The posterior is obtained by Eq. (22). The numerator terms are $6.001 \times 1.33 = 7.981$ for N_1 and $4.001 \times 0.66 = 2.640$ for N_2, and the denominator is their sum $Z = 7.981 + 2.640 = 10.621$. The final posterior satisfies $P(N_1 \mid T_1) = \frac{7.981}{10.621} \approx 0.751$, $P(N_2 \mid T_1) = \frac{2.640}{10.621} \approx 0.249$. It favors assigning task T_1 to the fast function instance N_1. A small probability for N_2 remains, which preserves exploration.

4.4 Variable neighborhood search strategy

The solution space of serverless MapReduce scheduling contains multiple local optima due to discrete task-to-instance assignments and heterogeneous function execution speeds. Previous studies have shown that fixed-scale neighborhood search is insufficient for such problems [36, 37], as small perturbations cannot escape local optima while large perturbations disrupt promising solutions. To address this limitation, a variable neighborhood search strategy is proposed that progressively adjusts perturbation strength based on search progress.

Algorithm 2 VNS_Optimize

Input: initial solution x_0, maximum neighborhood level $k_{\max}$
Output: optimized solution x_{best}

$x_{\text{best}} \leftarrow x_0$; $k \leftarrow 1$;
while $k \leqslant k_{\max}$ **do**
  $x' \leftarrow$ sample from neighborhood $\Omega_k(x_{\text{best}})$ by Eq. (25);
  $x'' \leftarrow$ local search on x';
  if $f(x'') < f(x_{\text{best}})$ **then**
    $x_{\text{best}} \leftarrow x''$;
    $k \leftarrow 1$ // reset to finest neighborhood;
  end if
  else
    $k \leftarrow k + 1$ // move to next neighborhood;
  end if
end while
return x_{best};

Algorithm 2 formalizes the VNS optimization procedure. The strategy employs three neighborhood operators with progressively increasing perturbation strength.

The first level applies swap-based perturbation. Two tasks are randomly selected, and their assigned function instances are exchanged. This operation modifies only two assignments, enabling fine-grained exploration of solutions near the current configuration.

The second level uses reassignment-based perturbation. A single randomly selected task is reassigned to a different

Just Accepted

function instance from its candidate set. This targets individual bottleneck tasks that may limit overall makespan.

The third level employs shuffle-based perturbation. A continuous sequence of 3–7 tasks is randomly selected, and their function instance assignments are shuffled. This large-scale restructuring enables exploration of distant solution regions when fine-grained adjustments fail to improve.

The hierarchical operator mapping is formalized as $\Phi(k)$, where the neighborhood $\Omega_k(x)$ is defined by

$$\Omega_k(x) = \{y \mid y = \Phi(k)(x)\} \tag{25}$$

At each iteration, the shaking stage samples a candidate solution from the current neighborhood. The local search phase then refines this sampled candidate using iterative hill-climbing. In local search, any improvement above to the sampled candidate is accepted immediately. The search is stopped after 20 consecutive attempts without improvement. The locally optimized solution is then compared with the global best solution. If it improves upon the global best, the search resets to the finest neighborhood for intensified exploration. Otherwise, the algorithm advances to the next neighborhood. The entire procedure terminates when all neighborhood levels have been explored without finding further improvements.

4.5 Dependency-aware crossover strategy

In serverless MapReduce, each reduce task depends on a subset of map tasks that must complete first. Standard two-point crossover may place reduce tasks before their map predecessors, producing infeasible schedules that violate execution dependencies. To solve this issue, a dependency-aware crossover strategy is proposed. The strategy fixes the topological order to respect all dependencies and applies crossover only to function instance assignments.

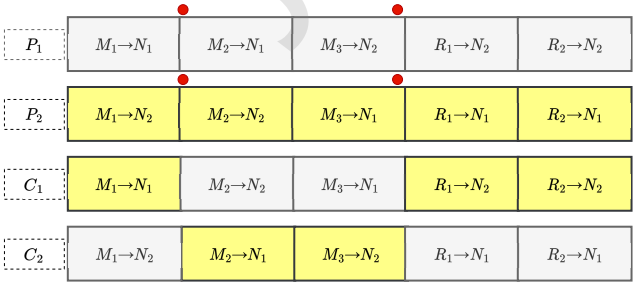


Fig. 6 Dependency-aware crossover strategy

The dependency-aware crossover operates in two steps.

1) Fixed topological order. A topological sort of the MapReduce DAG determines a fixed task sequence for all individuals in the population. This sequence respects all map-to-reduce dependencies.

2) Assignment-only crossover. Two-point crossover is applied only to the function instance assignment vector, keeping the task order fixed. Two crossover points are randomly selected. Assignments within the segment are inherited from one parent, while assignments outside inherit from the other parent.

Fig. 6 illustrates the key difference on a two-layer MapReduce job with tasks m_1, m_2, m_3, r_1, r_2 and dependencies $m_1, m_2 \to r_1$ and $m_2, m_3 \to r_2$. The fixed topological order is $[m_1, m_2, m_3, r_1, r_2]$. A standard crossover that swaps both task order and function instance assignments may produce $[m_1, r_1, m_2, m_3, r_2]$, where r_1 precedes its predecessor m_2, violating the dependency. In contrast, the dependency-aware strategy fixes this topological order and applies two-point crossover only to function instance assignments. With crossover segment $[m_2, m_3]$, both offspring inherit assignments inside the segment from one parent and outside from the other, while preserving the task sequence $[m_1, m_2, m_3, r_1, r_2]$. As shown, this mechanism ensures all offspring automatically satisfy dependency constraints without repair.

In summary, because task order always follows the topological sequence, all $M \to R$ dependencies hold and offspring can be decoded directly without repair. Each crossover operation generates two feasible offspring, one of which is retained and added to the next generation after mutation. This design enables efficient recombination of parental task-to-instance mappings while guaranteeing dependency feasibility.

5 Experiments

5.1 Experimental settings

To simulate MapReduce workflows such as log analysis at varying scales, two-layer DAGs were generated following the construction rules in [4]. The number of map and reduce tasks grows with the whole-job input size, which ranges from roughly 1 to 200 GB. A task's execution time depends on the assigned function instance. The reduce stage takes the map outputs as input; the total shuffle data size equals the sum of map outputs. The experiments involve five job types (Job0–Job4), with 10 instances per type, totaling 50 instances.

BGA was implemented in Java SE 8 and executed on a 64-bit Windows 10 system with an Intel® Core™ i7-12700 CPU at 2.10 GHz and 32 GB of RAM.

Performance is evaluated using Relative Percentage Deviation (RPD) [38]:

$$\text{RPD} = \frac{F - F^*}{F^*} \times 100 \tag{26}$$

Just Accepted

where F is the algorithm's makespan and F^* is the best makespan on the same instance. The best makespan F^* is taken across all runs or algorithms. Smaller RPD indicates better performance. Statistical significance is assessed using 95% LSD confidence intervals over repeated runs.

5.2 Model validation with CPLEX

To validate the mixed-integer programming model in Section 3, CPLEX is applied as an exact solver on instances with 7 to 56 tasks. BGA is independently run 10 times on each instance and the average result is shown; CPLEX is executed once per instance.

Table 2 shows that CPLEX obtains optimal solutions for instances Task7 through Task13. BGA attains the same or nearly the same makespan within a fixed 5 s budget: both algorithms yield 15.32 s on Task7, and BGA's gap on Task9 is only 0.26%. However, CPLEX CPU time increases from 0.81 s on Task7 to 333.06 s on Task13. For Task15 and larger instances, CPLEX fails to converge within a two-hour limit. In contrast, BGA successfully generates feasible schedules for all eight instances within 5 s time limit.

These results validate the proposed formulation on instances solved to optimality and demonstrate that BGA achieves near-optimal solutions efficiently. The increase in exact-solver cost with task scale is consistent with the NP-hardness of the problem and justifies the use of heuristic methods.

Table 2 Comparison between CPLEX and BGA on the validation set

Instance	CPLEX		BGA	
	Makespan(s)	Time(s)	Makespan(s)	Time(s)
Task7	**15.32**	0.81	**15.32**	5.00
Task9	**19.00**	3.63	19.05	5.00
Task11	**16.67**	89.88	**16.67**	5.00
Task13	**18.98**	333.06	**18.98**	5.00
Task15	–	–	**21.43**	5.00
Task28	–	–	**34.41**	5.00
Task42	–	–	**42.85**	5.00
Task56	–	–	**43.30**	5.00

5.3 Parameter calibration

Parameter settings significantly affect the performance of evolutionary algorithms. We calibrate three parameters using a full factorial experiment: population size $P_s \in \{2, 50, 100, 150\}$, VNS trigger interval $I_{\text{vns}} \in \{5, 10, 35, 60\}$, and Bayesian offspring ratio $R_b \in \{0.1, 0.3, 0.5, 0.7\}$. These parameters control population diversity, local search frequency, and the balance between Bayesian-guided and crossover-based offspring.

A full factorial design tests 64 combinations (4×4×4). Each combination is run 10 times on 10 instances, which is a total of 6,400 runs. Mean RPD is computed for each configuration to quantify overall performance. One-way ANOVA is performed for each parameter to assess the significance of its main effect. Fisher's LSD post-hoc tests at a 95% confidence level are then conducted to identify the optimal parameter values.

As shown in Fig. 7(a), mean RPD decreases as population size increases from 2 to 50, then stabilizes at larger sizes. Fig. 7(b) shows that $I_{\text{vns}} = 10$ outperforms other intervals. Fig. 7(c) indicates that $R_b = 0.3$ yields the lowest RPD. Therefore, the configuration $P_s = 50$, $I_{\text{vns}} = 10$, and $R_b = 0.3$ is adopted.

5.4 Ablation experiments

To assess the contribution of each component, three ablation variants are tested: Exp1 removes Bayesian probabilistic modeling, Exp2 removes variable neighborhood search, and Exp3 removes dependency-aware crossover. Each variant is run 10 independent times on instances ranging from Case 0 to Case 4, and effects are assessed, using the 95% LSD confidence interval. The complete BGA achieves consistently low and stable RPD values across all cases, with an average value of approximately 3%.

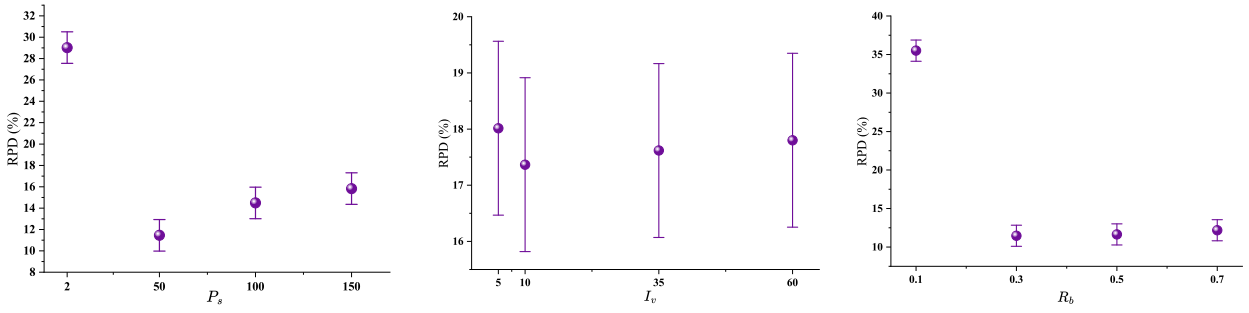
As shown in Fig. 8(a), Exp1 exhibits higher RPD across all cases, reaching about 107% in Case 4. This degradation indicates that Bayesian modeling is an efficient search guide in the nonlinear solution space. Fig. 8(b) shows that the RPD of Exp2 increases overall from around 60% to 100% with increasing complexity. This pattern indicates that variable neighborhood search is important to escape local optima in complex instances. As illustrated in Fig. 8(c), Exp3 remains at a relatively poor performance level, with RPD around 10% across all cases. This result demonstrates that dependency-aware crossover is essential for maintaining feasible and improvable solutions.

These results confirm that the three components are complementary: Bayesian modeling provides convergence direction, variable neighborhood search enables escape from local optima, and dependency-aware crossover ensures feasibility. Bayesian modeling is the most critical component, followed by variable neighborhood search and dependency-aware crossover.

5.5 Comparative experiments

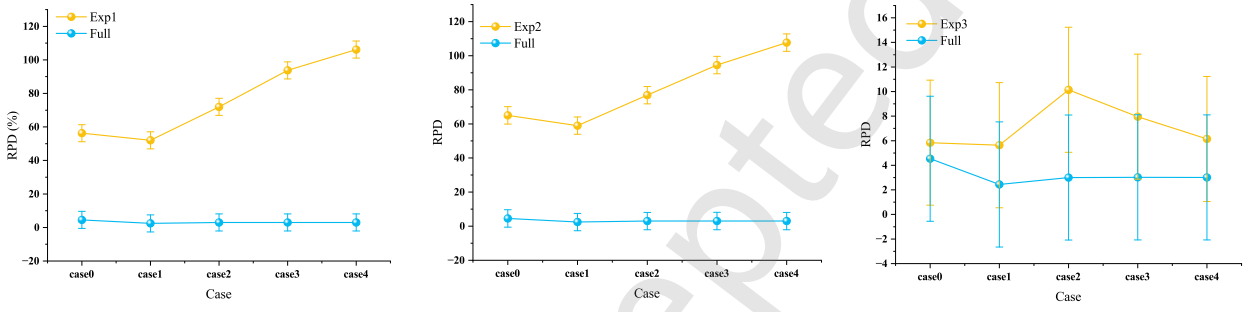
Five algorithms are compared: Slave Ants based Ant Colony Optimization (SACO) [39], Hybrid Grey Wolf Optimization

Just Accepted



a. Mean RPD under P_s b. Mean RPD under I_{vns} c. Mean RPD under R_b

Fig. 7 Parameter calibration



a. BGA vs. Exp1 b. BGA vs. Exp2 c. BGA vs. Exp3

Fig. 8 Ablation results

and Genetic Algorithm (GA-GWO) [40], Adaptive Particle Swarm Optimization (AdPSO) [41], Ant Colony Optimization based Bin Packing (ACO_BP) [18], and BGA. Each algorithm is independently run 10 times on each instance to ensure statistical reliability. Algorithm parameters are listed in Table 3, where N_a, N_w, N_p, and P_s denote the population sizes for different algorithms; p_c and p_m represent crossover and mutation rates; I_{vns} is the VNS trigger interval; R_b is the Bayesian offspring ratio; $a_{\max}$ is the maximum convergence factor; c is the coefficient vector; α and β are pheromone and heuristic weights; ρ and ϕ are evaporation rates; q_0 is the exploitation factor; $w_{\max}$ and $w_{\min}$ are the maximum and minimum inertia weights; c_1 and c_2 are cognitive and social factors. Parameters for comparative algorithms are taken from their original papers or common settings.

For fair comparison, all algorithms use the same CPU time budget of 5000 ms per run. The experiments involve five job types (Job0–Job4), with 10 instances per type, totaling 50 instances.

Table 4 shows mean makespan and RPD over 10 independent runs, while the corresponding standard deviations are shown in Table 5. BGA obtains the best makespan performance across all five job types, with values of 31.52, 132.86, 259.70, 398.07, and 519.59 s for Job0–Job4, respectively. Compared to the second-best method ACO_BP, BGA reduces

Table 3 Parameter settings of the compared algorithms

Algorithm	Parameters
ACO_BP	$N_a = 20, \alpha = 1, \beta = 2, \rho = 0.1$
AdPSO	$N_p = 20, w_{\max} = 0.9, w_{\min} = 0.4,$ $c_1 = 2.0, c_2 = 1.49455$
BGA	$P_s = 50, I_{\text{vns}} = 10, R_b = 0.3$
GA-GWO	$N_w = 200, a_{\max} = 2.0, p_c = 0.8,$ $p_m = 0.05, c = 0.5$
SACO	$N_a = 20, \alpha = 1.0, \beta = 2.0, \rho = 0.1,$ $q_0 = 0.8, \phi = 0.1$

makespan by about 26.9%, 19.3%, 22.3%, 24.4%, and 25.7% on Job0–Job4. BGA's RPD remains between 2.40% and 3.92% across all job types, consistently showing near-optimal performance. As shown in Fig. 9, BGA maintains a consistent performance advantage over all baseline algorithms as the problem scale increases.

Table 5 shows the standard deviations of RPD across Job0–Job4. BGA exhibits the best standard deviations ranging from 1.58% to 3.39%, indicating superior stability compared to baseline algorithms. In contrast, SACO shows the highest variability, suggesting that it has the poorest stability.

Table 6 presents Wilcoxon rank-sum test results comparing BGA against all baselines at significance level $\alpha = 0.05$. All comparisons yield $p < 0.001$ with large effect sizes of $r = 0.87$, confirming BGA's statistically significant and practically meaningful superiority.

Just Accepted

Table 4 Comparative experimental results

Case	ACO_BP		AdPSO		BGA		GA-GWO		SACO	
	Makespan(s)	RPD(%)	Makespan(s)	RPD(%)	Makespan(s)	RPD(%)	Makespan(s)	RPD(%)	Makespan(s)	RPD(%)
Job0	43.13	42.43	75.32	149.63	**31.52**	**3.92**	49.05	62.17	60.57	98.87
Job1	164.6	27.02	260.75	102.77	**132.86**	**2.79**	201.94	56.61	273.45	108.56
Job2	334.19	31.75	522.19	107.27	**259.7**	**2.61**	433.68	71.9	551.18	117.05
Job3	526.16	35.52	786.99	103.74	**398.07**	**2.69**	680.19	75.67	866.71	122.37
Job4	699.15	37.35	1036.97	105.24	**519.59**	**2.40**	933.85	84.53	1290.56	151.86

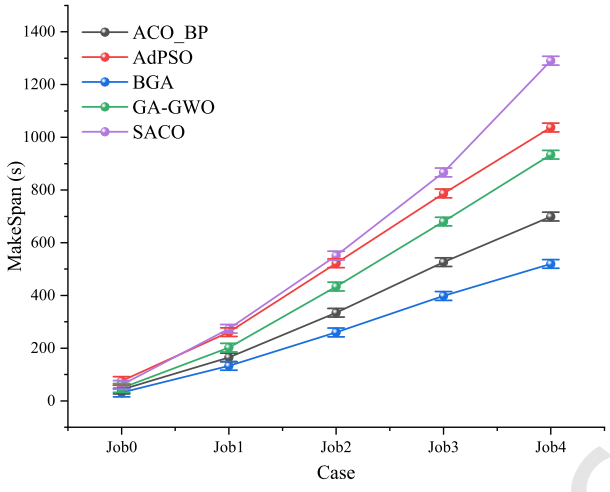


Fig. 9 Makespan of five algorithms on Job0–Job4

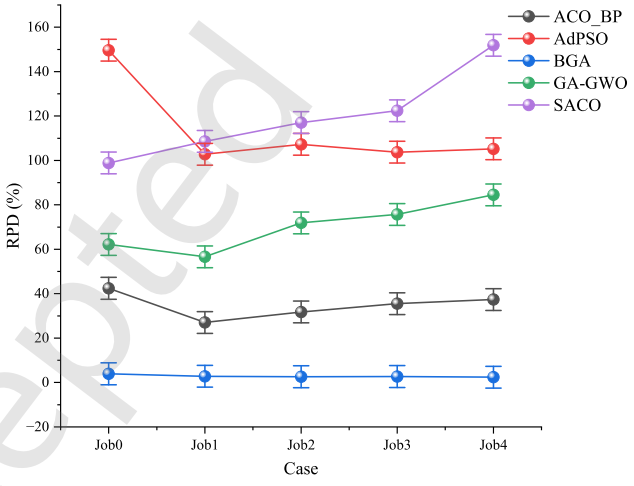


Fig. 10 RPD of five algorithms on Job0–Job4

Table 5 Standard deviations of RPD (%)

Case	ACO_BP	AdPSO	BGA	GA-GWO	SACO
Job0	3.91	23.73	**3.39**	14.43	19.61
Job1	4.29	18.49	**1.85**	11.23	40.96
Job2	5.28	18.26	**1.64**	13.74	33.81
Job3	4.11	14.38	**1.77**	10.12	51.93
Job4	6.59	14.67	**1.58**	13.88	50.18

Table 6 Wilcoxon rank-sum test results

Comparison	Z statistic	p-value	Effect size (r)
BGA vs. ACO_BP	-19.375	< 0.001	0.87
BGA vs. AdPSO	-19.375	< 0.001	0.87
BGA vs. GA-GWO	-19.375	< 0.001	0.87
BGA vs. SACO	-19.375	< 0.001	0.87

Fig. 10 shows the RPD trends across job types. BGA achieves the lowest RPD on all jobs, with values ranging from 2.40% to 3.92%, while baselines show larger deviations, especially on Job2–Job4. These results confirm that BGA achieves superior performance on both makespan and stability as problem complexity increases.

5.6 Performance advantage and mechanism analysis

Fig. 11 shows the convergence curves on Job1 over a 5000 ms runtime. BGA shows faster convergence in the early stage and maintains the lowest makespan after the initial search phase, confirming that Bayesian-guided sampling effectively

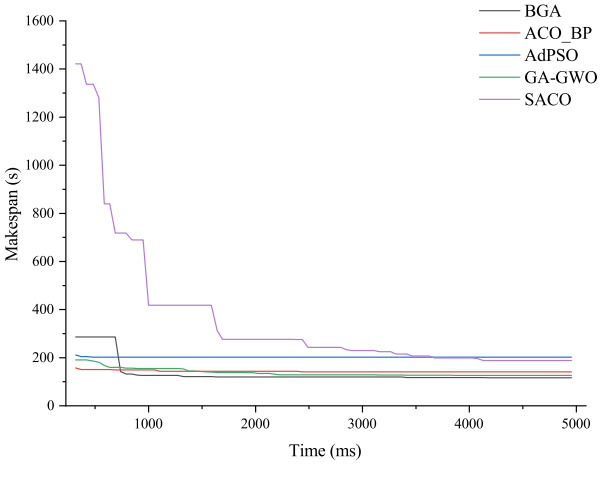


Fig. 11 Convergence comparison of all algorithms on Job1

directs the search toward high-quality solutions.

The experimental results show that BGA achieves the best makespan and the lowest RPD across Job0–Job4, with consistent gains over the second-best method (ACO_BP) and lower deviation than all baselines as job scale increases. The performance of BGA can be attributed to three mechanisms.

1) Stage-aware serverless MapReduce scheduling model. BGA formulates a mixed-integer programming model that explicitly captures MapReduce stage dependencies and their effect on scheduling, providing a unified basis for objectives

Just Accepted

and feasibility checks.

2) Bayesian-guided evolutionary search. BGA embeds a Bayesian probability model into genetic evolution to guide task-to-instance assignment. The model learns patterns from elite solutions and incorporates performance priors. This balances exploitation and exploration, keeping RPD low across all jobs.

3) Operator design with variable neighborhood search and dependency-aware crossover. Variable neighborhood search applies progressive perturbations to escape local optima under stagnation. Dependency-aware crossover preserves topological order during recombination, confining evolution to the feasible space and improving search efficiency.

In summary, BGA combines stage-aware modeling to structure the optimization, Bayesian guidance to provide stable convergence, and VNS with dependency-aware crossover to maintain feasible exploration. This synergy reduces both makespan and RPD and preserves the advantage as job size increases. The experimental data and mechanism analysis support the effectiveness of BGA for heterogeneous and dependency-coupled serverless MapReduce scheduling.

6 Conclusion

This study addresses the MapReduce scheduling problem under serverless resource constraints, aiming to minimize makespan while satisfying precedence dependencies and stateless execution constraints. To solve this problem, BGA is designed to accommodate its specific properties.

Comprehensive comparisons with CPLEX and state-of-the-art algorithms prove BGA's superiority in both solution quality and computational efficiency. The Bayesian probability model in BGA learns high-frequency task-to-instance assignment patterns from elite solutions and guides offspring generation through probabilistic sampling. By using dependency-aware crossover, it maintains precedence constraints during recombination and preserves solution feasibility. Variable neighborhood search helps the algorithm escape local optima through multi-scale perturbations. Finally, the integrated design of BGA effectively balances exploration and exploitation, and achieves robust performance across different problem scales.

This approach targets serverless MapReduce scheduling where makespan is the primary optimization objective. This work focuses on offline scheduling optimization, assuming batch task arrival and successful task completion. Our future research will include task failure scenarios and online scheduling with dynamic task arrivals. Inspired by recent advances in using large language models for algorithm design [42–44], we will explore LLM-based approaches to address emerging challenges in uncertain and dynamic scheduling scenarios.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (No. 62473186) and the Natural Science Foundation of Shandong Province (No. ZR2024MF017).

Reference

[1] N. Mahmoudi and H. Khazaei, Performance modeling of serverless computing platforms, *IEEE Transactions on Cloud Computing*, vol. 10, no. 4, pp. 2834–2847, 2022.

[2] X. Huang, R. Gu, and Y. Huang, Towards efficient serverless mapreduce computing on cloud-native platforms, *Big Data Mining and Analytics*, vol. 8, no. 3, pp. 575–591, 2025.

[3] F. Li, B. C. Ooi, M. T. Özsu, and S. Wu, Distributed data management using mapreduce, *ACM Comput. Surv.*, vol. 46, no. 3, pp. 31:1–31:42, 2014.

[4] C.-H. Chen, J.-W. Lin, and S.-Y. Kuo, Mapreduce scheduling for deadline-constrained jobs in heterogeneous cloud computing systems, *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 127–140, 2018.

[5] X. Tang, W. Cao, H. Tang, T. Deng, J. Mei, Y. Liu, C. Shi, M. Xia, and Z. Zeng, Cost-efficient workflow scheduling algorithm for applications with deadline constraint on heterogeneous clouds, *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 9, pp. 2079–2092, 2022.

[6] M. J. Nadjafi-Arani, S. Doostali, and M. Younis, Workflow scheduling with guaranteed responsiveness and minimal cost, *IEEE Transactions on Services Computing*, vol. 16, no. 4, pp. 3075–3087, 2023.

[7] M. S. Aslanpour, A. N. Toosi, M. A. Cheema, and M. B. Chhetri, Faashouse: Sustainable serverless edge computing through energy-aware resource scheduling, *IEEE Transactions on Services Computing*, vol. 17, no. 4, pp. 1533–1547, 2024.

[8] P. Sowkuntla and P. Sai Prasad, Parallel attribute reduction in high-dimensional data: An efficient mapreduce strategy with fuzzy discernibility matrix, *Applied Soft Computing*, vol. 172, p. 112870, 2025.

[9] H. Zheng and J. Wu, Joint scheduling of overlapping mapreduce phases: Pair jobs for optimization, *IEEE Transactions on Services Computing*, vol. 14, no. 5, pp. 1453–1463, 2021.

[10] H. Guo, Z. Miao, J. C. Ji, and Q. Pan, An effective collaboration evolutionary algorithm for multi-robot task allocation and scheduling in a smart farm, *Knowledge-Based Systems*, vol. 289, p. 111474, 2024.

[11] Z. Miao, H. Guo, Q.-k. Pan, C. Peng, and Z. Xu, A reinforcement learning-enhanced multi-objective iterated greedy algorithm for weeding-robot operation scheduling problems, *Expert Systems with Applications*, vol. 263, p. 125760, 2025.

[12] H. Guo, H. Sang, and Q. Pan, A learning-driven iterated local search algorithm for solving distributed flowshop robust scheduling problems, *Control and Decision*, vol. 41, no. 4, pp. 977–986, 2026.

Just Accepted

[13] H. Guo, Q. Pan, H. Sang, Z. Miao, and W. Zhang, Ahllns: An automated algorithm for multi-objective heterogeneous agricultural robot operation scheduling problems, *IEEE Transactions on Automation Science and Engineering*, vol. 23, pp. 5096–5109, 2026.

[14] C. Sun, H. Sang, L. Meng, B. Zhang, and T. Meng, Efficient multi-start gray wolf optimization algorithm for the distributed permutation flowshop scheduling problem with preventive maintenance, *Complex System Modeling and Simulation*, vol. 5, no. 2, pp. 107–124, June 2025.

[15] P. Wang, R. De Leone, and H. Sang, Improved dynamic q-learning algorithm to solve the lot-streaming flowshop scheduling problem with equal-size sublots, *Complex System Modeling and Simulation*, vol. 4, no. 3, pp. 223–235, September 2024.

[16] R. Zhang, H. Yu, A. S lowik, and K. Gao, Q-learning based meta-heuristics for scheduling bi-objective surgery problems with setup time, *Complex System Modeling and Simulation*, vol. 4, no. 4, pp. 321–338, December 2024.

[17] C. Chen, H. Sang, H. Guo, and M. Wang, Dependency-aware and dynamic-repair multi-objective optimization for software projects, *Control Theory & Applications*, pp. 1–9, 2026, advance online publication. [Online]. Available: https://link.cnki.net/urlid/44.1240.tp.20260913.1951.006

[18] R. Jeyaraj and A. Paul, Optimizing mapreduce task scheduling on virtualized heterogeneous environments using ant colony optimization, *IEEE Access*, vol. 10, pp. 55 842–55 855, 2022.

[19] U. Demirbaga, Ecocloud: Green computing through energy and carbon efficient task scheduling in industrial iot-enabled cloud environments, *IEEE Internet of Things Journal*, vol. 12, no. 17, pp. 34 644–34 652, Sept. 2025.

[20] C. Diwaker, V. Hasanpuri, Y. Gulzar, and B. Sharma, Optimizing mapreduce efficiency and reducing complexity with enhanced particle swarm optimization (mr-mpso), *Swarm and Evolutionary Computation*, vol. 95, p. 101917, 2025.

[21] M. Bergui, S. Hourri, S. Najah, and N. S. Nikolov, Predictive modelling of mapreduce job performance in cloud environments using machine learning techniques, *Journal of Big Data*, vol. 11, no. 1, p. 98, 2024.

[22] A. Mampage, S. Karunasekera, and R. Buyya, A deep reinforcement learning based algorithm for time and cost optimized scaling of serverless applications, *Future Generation Computer Systems*, vol. 173, p. 107873, 2025.

[23] Y. Chen, B. Liu, W. Lin, Y. Guo, and Z. Peng, Casr: Optimizing cold start and resources utilization in serverless computing, *Future Generation Computer Systems*, vol. 170, p. 107851, 2025.

[24] Y. Zeng, J. Zhou, B. Ye, Z. Qu, S. Guo, T. Gong, and P. Li, Expertdrl: Request dispatching and instance configuration for serverless edge inference with foundation models, *IEEE Transactions on Mobile Computing*, vol. 24, no. 9, pp. 8089–8104, 2025.

[25] M. Ghorbian, M. Ghobaei-Arani, and L. Esmaeili, Autonomous scheduling mechanism based on energy awareness for improving resource allocation in serverless iot edge, *Scientific Reports*, vol. 15, no. 1, p. 24173, 2025.

[26] P. Wang, Y. Li, C. Fang, Y. Zhong, and Z. Ding, Optimizing serverless performance through game theory and efficient resource scheduling, *IEEE Transactions on Computers*, vol. 74, no. 6, pp. 1990–2002, 2025.

[27] B. Feng, Z. Ding, and C. Jiang, Heterogeneity-aware proactive elastic resource allocation for serverless applications, *IEEE Transactions on Services Computing*, vol. 17, no. 5, pp. 2473–2487, 2024.

[28] J. Zhang, X. Li, L. Chen, and R. Ruiz, Scheduling workflows with limited budget to cloud server and serverless resources, *IEEE Transactions on Services Computing*, vol. 17, no. 4, pp. 1766–1779, 2024.

[29] D. Cheng, K. Yan, X. Cai, Y. Gong, and C. Hu, Slo-aware function placement for serverless workflows with layer-wise memory sharing, *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 6, pp. 1074–1091, 2024.

[30] B. Zolfaghari, S. Abrishami, A. Rasoolzadegan, and B. Javadi, Dynamic function placement and request scheduling of serverless workflows in edge environment, *IEEE Transactions on Services Computing*, pp. 1–13, 2025.

[31] J. Jarachanthan, L. Chen, F. Xu, and B. Li, *Astrea*: Auto-serverless analytics towards cost-efficiency and qos-awareness, *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 3833–3849, Dec. 2022.

[32] C. Chen, H. Sang, J. Zhang, H. Liu, and H. Chi, Efficient serverless resource allocation for mapreduce job based on a* algorithm, in *2024 International Conference on New Trends in Computational Intelligence (NTCI)*, Oct. 2024, pp. 212–216.

[33] S. Zhou, X. He, Z. Zhang, F. Gao, Y. Hong, R. Gao, X. Wang, H. Yun, R. Ma, and J. Sun, Stardata: Distributed map-reduce framework atop serverless computing platforms, *IEEE Access*, vol. 13, pp. 182 633–182 644, 2025.

[34] X. Li, F. Chen, R. Ruiz, and J. Zhu, Mapreduce task scheduling in heterogeneous geo-distributed data centers, *IEEE Transactions on Services Computing*, vol. 15, no. 6, pp. 3317–3329, 2022.

[35] Y. Lu, L. Yang, D. Xia, F. Meng, and S. Sharif Azadeh, Incorporating reservation strategies into demand management and train scheduling in metro systems, *Transp. Res. Part C Emerg. Technol.*, vol. 182, p. 105441, Jan. 2026.

[36] M. I. Khaleel, A dynamic weight–assignment load balancing approach for workflow scheduling in edge-cloud computing using ameliorated moth flame and rock hyrax optimization algorithms, *Future Generation Computer Systems*, vol. 155, pp. 465–485, 2024.

[37] X. Zhang and X. Wu, Multi-objective variable neighborhood search algorithm to optimize the multi-product u-shaped disassembly line balancing problem with a fixed number of stations, *Complex System Modeling and Simulation*, vol. 5, no. 4, pp. 354–369, December 2025.

[38] H. Sang, Z. Li, and M. F. Tasgetiren, An effective optimization

Just Accepted

method for integrated scheduling of multiple automated guided vehicle problems, *Tsinghua Science and Technology*, vol. 29, no. 5, pp. 1355–1367, Oct. 2024.

[39] Y. Moon, H. Yu, J.-M. Gil, and J. Lim, A slave ants based ant colony optimization algorithm for task scheduling in cloud computing environments, *Hum.-Centric Comput. Inf. Sci.*, vol. 7, no. 1, p. 28, Oct. 2017.

[40] I. Behera and S. Sobhanayak, Task scheduling optimization in heterogeneous cloud computing environments: A hybrid ga-gwo approach, *Journal of Parallel and Distributed Computing*, vol. 184, p. 104739, Feb. 2024.

[41] S. Nabi, M. Ahmad, M. Ibrahim, and H. Hamam, Adpso: Adaptive pso-based task scheduling approach for cloud computing, *Sensors*, vol. 22, no. 3, Jan. 2022, art. no. 920.

[42] F. Liu, Y. Yao, P. Guo, Z. Yang, X. Lin, Z. Zhao, X. Tong, K. Mao, Z. Lu, Z. Wang, M. Yuan, and Q. Zhang, A systematic survey on large language models for algorithm design, *ACM Computing Surveys*, vol. 58, no. 8, pp. 218:1–218:32, 2026.

[43] S. Yao, F. Liu, X. Lin, Z. Lu, Z. Wang, and Q. Zhang, Multi-objective evolution of heuristic using large language model, *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 25, pp. 27 144–27 152, 2025.

[44] F. Wang, J. Chen, Y. Du, Y. Song, Y. Chen, R. Mallipeddi, and W. Pedrycz, Llm-assisted adaptive large neighborhood search for agile earth observation satellite scheduling, *Engineering Management*, vol. 13, no. 1, pp. 213–239, 2026.

Author biography



Chuanzhi Chen received the BEng degree from Liaocheng University, Liaocheng, China, in 2024. He is currently pursuing the MEng degree in software engineering at Liaocheng University, Liaocheng, China. His current research interests include cloud computing.



Hongyan Sang received the MS degree from Liaocheng University, Liaocheng, China, in 2010, and the PhD degree in industrial engineering from Huazhong University of Science and Technology, Wuhan, China, in 2013. Since 2003, she has been with the School of Artificial Intelligence and Computer Science, Liaocheng University, Liaocheng, China, where she became a professor in 2021. She has authored more than 100 refereed papers. Her current research interests include intelligent optimization and scheduling. She is a Senior Member of IEEE. She is Senior Editor of Swarm and Evolutionary Computation and Associate Editor of Expert Systems with Applications and Complex System Modeling and Simulation.



Biao Zhang is currently an associate professor in the School of Artificial Intelligence and Computer Science at Liaocheng University, China. He received the B.S. and Ph.D. degrees from Shandong University of Technology, Zibo, China and Huazhong University of Science and Technology, Wuhan, China, in 2012 and 2019, respectively. His current research interests include machine learning and intelligent optimisation.



Jinquan Zhang received his Ph.D. degree in the College of Computer Science and Engineering from Southeast University, Nanjing, China, in 2024. He is currently a lecturer in the School of Computer Science and Technology, Guangdong University of Technology, Guangzhou, China. He is the author or co-author of about 10 papers in international journals and conferences, such as IEEE Transactions on Services Computing, IEEE Transactions on Cloud Computing, and Future Generation Computer Systems. His research interests focus on cloud resource management and serverless computing.



Hengwei Guo received the B.S. degree in computer science from Liaocheng University, Liaocheng, China, in 2022, and the Ph.D. degree from the School of Mechatronic Engineering and Automation, Shanghai University, Shanghai, China, in 2026. His current research interests focus on computational intelligence, specifically reinforcement learning, evolutionary computation, and automatic algorithm design for multi-agent task allocation and intelligent scheduling.

Just Accepted