APPENDIX

A SYSTEM-LEVEL DESIGN OBJECTIVES

In resource-constrained, heterogeneous environments, designing a distributed system for graph computation requires clear, measurable goals. We introduce four core system-level metrics that guide Zsiga's design and evaluation: execution time, communication volume, memory usage, and resource-aware load imbalance. These metrics are selected to capture the computational, communication, and scalability challenges of connected component computation on heterogeneous servers.

Execution Time (T_{total}). We measure the total wall-clock time to complete the end-to-end computation, including the initial partitioning, iterative communication and computation, and the final halting step. Formally,

$$T_{\text{total}} = T_{\text{partition}} + \sum_{r=1}^{R} (T_{\text{comm}}^{(r)} + T_{\text{compute}}^{(r)}) + T_{\text{halt}},$$

where R is the number of iterations until convergence, $T_{\text{comm}}^{(r)}$ and $T_{\text{compute}}^{(r)}$ represent the communication and computation time in iteration r, and T_{halt} is the final computation cost after convergence.

Communication Volume ($|\mathcal{M}|$). To quantify the synchronization cost, we track the total number of inter-server messages exchanged during all refinement rounds. Each server i sends edge updates of the form $(u, p(u))$ to both $\xi(u)$ and $\xi(p(u))$ whenever the parent pointer of u changes across server boundaries. We define:

$$|\mathcal{M}| = \sum_{r=1}^{R} \sum_{i=1}^{\rho} |\mathcal{M}_i^{(r)}|,$$

where $\mathcal{M}_i^{(r)}$ is the set of messages sent by server i in iteration r. This metric captures the network cost of maintaining consistency across distributed partitions and is bounded theoretically in Theorem 4.4.

Load Imbalance ($\delta_{imbalance}$). To capture workload skew under heterogeneity, we define load imbalance as the difference between the maximum and minimum per-server computation time, normalized by the average:

$$\delta_{\text{imbalance}} = \frac{\max_i T_{\text{compute}}(i) - \min_i T_{\text{compute}}(i)}{\frac{1}{\rho} \sum_{j=1}^{\rho} T_{\text{compute}}(j)},$$

where $T_{\text{compute}}(i)$ denotes the cumulative computation time on server i. High imbalance indicates poor adaptation to server capacity and may lead to underutilized or bottlenecked servers.

Memory Usage (M_i). Each server i maintains a subgraph (V_i, E_i) and a hash table for parent pointers during Union-Find. Assuming proportional partitioning, we measure:

$$M_i = |V_i| + |E_i|,$$

As shown in Theorem 4.3, the expected bound is $O((|V| + |E|)/\rho)$, assuming a reasonably balanced partitioning scheme.

B PROOFS OF THEORETICAL RESULTS

In this appendix, we provide the detailed proofs of the lemmas and theorems stated in the main text. These results support the claims made in Section 4.2 and 4.3, particularly concerning partitioning balance, edge-cut bounds, and memory usage.

B.1 Proof of Lemma 4.1

Lemma. For any server i, the number of assigned vertices satisfies $\frac{R_i}{R_{\text{total}}} \cdot |V| - 1 \leq \{u \in V | \xi(u) = i\} \leq \frac{R_i}{R_{\text{total}}} \cdot |V| + 1$, where R_i is the resource capacity of server i and R_{total} is the total capacity across all servers.

Proof. Let $W[i] = \frac{R_i}{R_{\text{total}}} \cdot |V|$ represent the ideal number of vertices assigned to server i based on its share of the total resource capacity. The partitioning algorithm aims to approximate this target while considering both resource proportionality and graph locality. Since vertices are assigned one by one and allocations must be discrete, rounding effects may cause a slight deviation from the ideal workload. In the worst case, this deviation is bounded by at most one vertex per server. Thus, the actual number of assigned vertices satisfies: $\frac{R_i}{R_{\text{total}}} \cdot |V| - 1 \leq \{u \in V | \xi(u) = i\} \leq \frac{R_i}{R_{\text{total}}} \cdot |V| + 1$. Substituting the expression for $W[i]$, we obtain the desired bound. □

B.2 Proof of Lemma 4.2

Lemma. The expected number of edge cuts by the partitioning algorithm is bounded by $|E_{\text{cut}}| \leq |E| - \sum_{u \in V} d(u, \xi(u))$, where $d(u, \xi(u))$ is the number of neighbors of vertex u assigned to the same server $\xi(u)$, and $|E_{\text{cut}}|$ is the number of edges crossing server boundaries.

Proof. Let $|E|$ denote the total number of edges in the graph, and let $|E_{\text{cut}}|$ be the number of edges whose endpoints are assigned to different servers. For each vertex $u \in V$, define $d(u, \xi(u))$ as the number of neighbors of u that are also assigned to server $\xi(u)$. Summing $d(u, \xi(u))$ across all vertices gives the total number of intra-server neighbor connections, counted from one side of each edge. Therefore, the total number of intra-server edges is at least $\frac{1}{2} \sum_{u \in V} d(u, \xi(u))$, and the number of cut edges is bounded above by $|E_{\text{cut}}| = |E| - (\text{number of intra - server edges}) \leq |E| - \sum_{u \in V} d(u, \xi(u))$.

The partitioning algorithm aims to maximize $\sum_{u \in V} d(u, \xi(u))$ by assigning each vertex u to a server where many of its neighbors have already been assigned. Processing high-degree vertices first increases the chances that subsequent lower-degree neighbors can align to the same server, further reducing the number of edge cuts. Since $d(u, \xi(u)) \leq d(u)$ for all u, the bound follows directly. □

B.3 Proof of Theorem 4.3

Theorem. Each server in Zsiga uses at most $O((|V| + |E|)/\rho)$ memory in expectation, where $|V|$ and $|E|$ are the number of vertices and edges in the graph, and ρ is the number of servers.

Proof. Zsiga's partitioning algorithm assigns vertices and their incident edges to servers proportionally to their computational resources (e.g., CPU, memory, and bandwidth). Let server i have a resource capacity R_i, and let the total capacity be $R_{\text{total}} = \sum_{j=1}^{\rho} R_j$.

The number of vertices and their incident edges assigned to server i is approximately $|V_i| \approx \frac{R_i}{R_{\text{total}}} \cdot |V|$, $|E_i| \approx \frac{R_i}{R_{\text{total}}} \cdot |E|$. Since $R_i/R_{\text{total}} \leq 1$ and the maximum value occurs when all servers are homogeneous (i.e., $R_i = R_{\text{total}}/\rho$), we get $|V_i| + |E_i| = O\left(\frac{|V|+|E|}{\rho}\right)$. Thus, each server maintains a proportionally balanced subset of vertices and edges, resulting in bounded memory usage of $O((|V|+|E|)/\rho)$. □

B.4 Proof of Theorem 4.4

Theorem. Upon convergence, Zsiga requires at most $C \times (\rho - 1)$ messages across servers, where C is the number of connected components and ρ is the number of servers. Therefore, the total network communication is bounded by $O(C \times (\rho - 1))$.

Proof. After convergence, each vertex u either points to its local root $r_{\text{local}} \in \xi(u)$ or, if it is a local root, to the global root of its connected component. Only local roots may send their parent pointers across servers. In each connected component, there is exactly one global root and at most $\rho - 1$ local roots—one per server, excluding the server hosting the global root. Each of these local roots sends an edge to the global root's server. Thus, each connected component incurs at most $\rho - 1$ inter-server messages. Summing across all C connected components yields a total of at most $C \times (\rho - 1)$ cross-server messages. Hence, the total network communication is $O(C \times (\rho - 1))$. □

C DETAILED ALGORITHMS

This section provides the complete pseudocode listings for Zsiga's core algorithms, including the overall workflow, adaptive resource-aware partitioning, network communication, edge refinement, and path compression.

C.1 Overall Workflow of Zsiga

Algorithm 1 Zsiga

Input: Undirected graph $G = (V, E)$
Output: $\{(u, m(\Lambda(u, G)))|u \in V\}$
$\xi \leftarrow$ AdaptiveResourceAwarePartitioning(G, $\{R_i\}_{i=1}^{\rho}$, ρ) ▷ Algorithm 2
repeat
  $\{E_i'\}_{i=1}^{\rho} \leftarrow$ NetworkCommunication($\{G_i\}_{i=1}^{\rho}$, $\{p(u, G_i)\}$) ▷ Algorithm 3
  EdgeRefinement($\{G_i\}_{i=1}^{\rho}$, $\{E_i'\}_{i=1}^{\rho}$) ▷ Algorithm 4
until no parent pointer changes on any server
PathCompression($\{G_i\}_{i=1}^{\rho}$) ▷ Algorithm 5

C.2 Adaptive Resource-Aware Partitioning

Algorithm 2 Adaptive Resource-Aware Partitioning

Input: Undirected graph $G = (V, E)$, resource capacities $\{R_i\}_{i=1}^{\rho}$, number of servers ρ
Output: Vertex-to-server mapping $\xi : V \to \{1, 2, \ldots, \rho\}$
Compute total capacity: $R_{\text{total}} = \sum_{i=1}^{\rho} R_i$
Compute proportional workloads: $W[i] = \frac{R_i}{R_{\text{total}}} \cdot |V| \; \forall i$
Initialize assigned workloads: Assigned$[i] \leftarrow 0$, $\forall i$
Order vertices V by descending degree $d(v)$
for each vertex $u \in V$ (in order) **do**
  Find server* = $\arg\max_i \{W[i] -$ Assigned$[i] \mid u$ has a neighbor assigned to server $i\}$
  if server* exists and Assigned$[i] < W[i]$ **then**
    $\xi(u) \leftarrow$ server* ▷ Assign u to server*
    Assigned[server*] $\leftarrow$ Assigned[server*] + 1
    ▷ Update assigned workload
  else
  ▷ Assign u to the server with the highest remaining workload capacity
    server* = $\arg\max_i \{W[i] -$ Assigned$[i]\}$
    $\xi(u) \leftarrow$ server*
    Assigned[server*] $\leftarrow$ Assigned[server*] + 1
Return: ξ

C.3 Network Communication

Algorithm 3 Network Communication

for servers $i \leftarrow 0$ to $\rho - 1$ **in parallel do**
  Let $G_i = (V_i, E_i)$ be the graph stored on server i
  for each vertex $u \in V_i$ **do**
    if $p(u, G_i)$ is changed by edge refinement or $u = m([\Lambda(u, G_i)]_{\xi(u)})$ **then**
      $v \leftarrow p(u, G_i)$
      Send (u, v) to servers $\xi(u)$ and $\xi(v)$

C.4 Edge Refinement

Algorithm 4 Edge Refinement

function ConnectedComponentComputation
  for all servers $i \leftarrow 0$ to $\rho - 1$ **in parallel do**
    Let $G_i = (V_i, E_i)$ be the subgraph assigned to server i
    Let E_i' be the edges received by server i via network
    Union-Find(E_i', G_i) ▷ Integrate E_i' into G_i
    Refinement(G_i) ▷ Adjust edges for load balancing
function Union-Find$(E^\star, G')$
  for all edges $(u, v) \in E^\star$ **do**
    $r_u \leftarrow$ FindRoot(u, G')
    $r_v \leftarrow$ FindRoot(v, G')
    if $r_u < r_v$ **then**
      $p(r_v, G') \leftarrow r_u$
    else
      $p(r_u, G') \leftarrow r_v$
  for all vertices $u \in V'$ **do**
    $r \leftarrow$ FindRoot(u, G')
    while $u \neq r$ **do**
      $t \leftarrow p(u, G')$
      $p(u, G') \leftarrow r$
      $u \leftarrow t$
function FindRoot(u, G')
  while $u \neq p(u, G')$ **do**
    $u \leftarrow p(u, G')$
  return u
function Refinement(G')
  for all vertices $u \in V'$ **do**
    if $u \neq m([\Lambda(u, G')]_{\xi(u)})$ **then**
      $p(u, G') \leftarrow m([\Lambda(u, G')]_{\xi(u)})$
    else
      $p(u, G') \leftarrow m(\Lambda(u, G'))$

C.5 Path Compression

Algorithm 5 Path Compression

for each server $i \leftarrow 0$ to $\rho - 1$ **in parallel do**
  Let $G_i = (V_i, E_i)$ be the subgraph stored on server i
  for each vertex $u \in V_i$ **do**
    $r \leftarrow$ FindRoot(u, G_i) ▷ In Algorithm 4
    $a \leftarrow u$
    while $a \neq r$ **do**
      $t \leftarrow p(a, G_i)$
      $p(a, G_i) \leftarrow r$
      $a \leftarrow t$