GIANT: Experiments

Settings

- Solve the ℓ_2-regularized logistic regression:

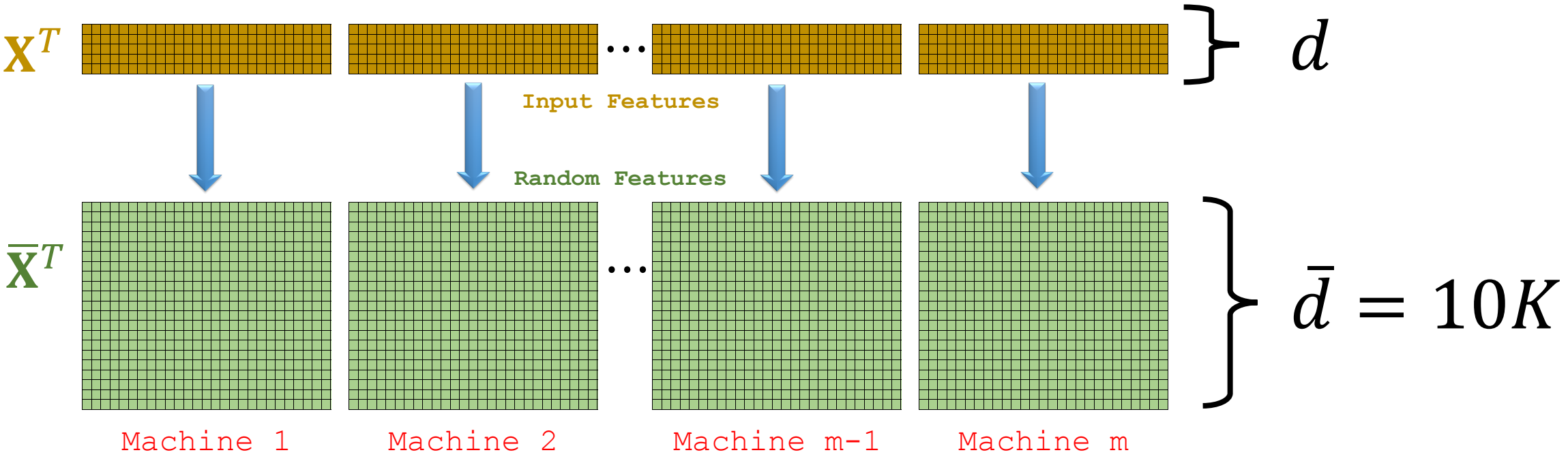
$$\min_{\mathbf{w}\in\mathbb{R}^d} \quad \left\{ f(\mathbf{w}) \triangleq \frac{1}{n}\sum_{j=1}^{n} \log\left(1+e^{-y_j \mathbf{x}_j^T \mathbf{w}}\right) + \frac{\gamma}{2}\|\mathbf{w}\|_2^2 \right\}$$

Datasets

- Covtype: $n = 581\text{K}, d = 54$.
- Epsilon: $n = 500\text{K}, d = 2\text{K}$.
- 80% for training, 20% for test.

Datasets

- Covtype: $n = 581\text{K}, d = 54$.
- Epsilon: $n = 500\text{K}, d = 2\text{K}$.
- 80% for training, 20% for test.



Compared Methods

- Accelerated gradient descent (AGD)
  - choose *step size* from {0.1, 1, 10, 100}
  - choose *momentum* from {0.5, 0.9, 0.95, 0.99, 0.999}

Compared Methods

- Accelerated gradient descent (AGD)
- Limited memory BFGS (a quasi-Newton method)
  - choose *number of history* from {30, 100, 300}
  - line search is used

Compared Methods

- Accelerated gradient descent (AGD)
- Limited memory BFGS
- DANE (another Newton-type method) [Shamir et al. 2014]
  - local solver: SVRG (a stochastic optimization method)
  - choose *step size of SVRG* from {0.1, 1, 10, 100}
  - choose *max. iteration of SVRG* from {30, 100, 300}

Reference:

Shamir, Srebro, & Zhang. Communication Efficient Distributed Optimization using an Approximate Newton-type Method. In *ICML*, 2014.

Compared Methods

- Accelerated gradient descent (AGD)
- Limited memory BFGS
- DANE (another Newton-type method)
- GIANT
  - local solver: conjugate gradient (CG)
  - choose *max iteration of CG* from {30, 100, 300}

Compared Methods

- Accelerated gradient descent (AGD) — 2 Tuning Parameters
- Limited memory BFGS — 1 Tuning Parameter
- DANE (another Newton-type method) — 2 Tuning Parameters
- GIANT — 1 Tuning Parameter

Experiment Environment

- Spark 2.1.1 + Scala 2.11.8

Experiment Environment

- Spark 2.1.1 + Scala 2.11.8
- Cori Supercomputer (Cray XC40)

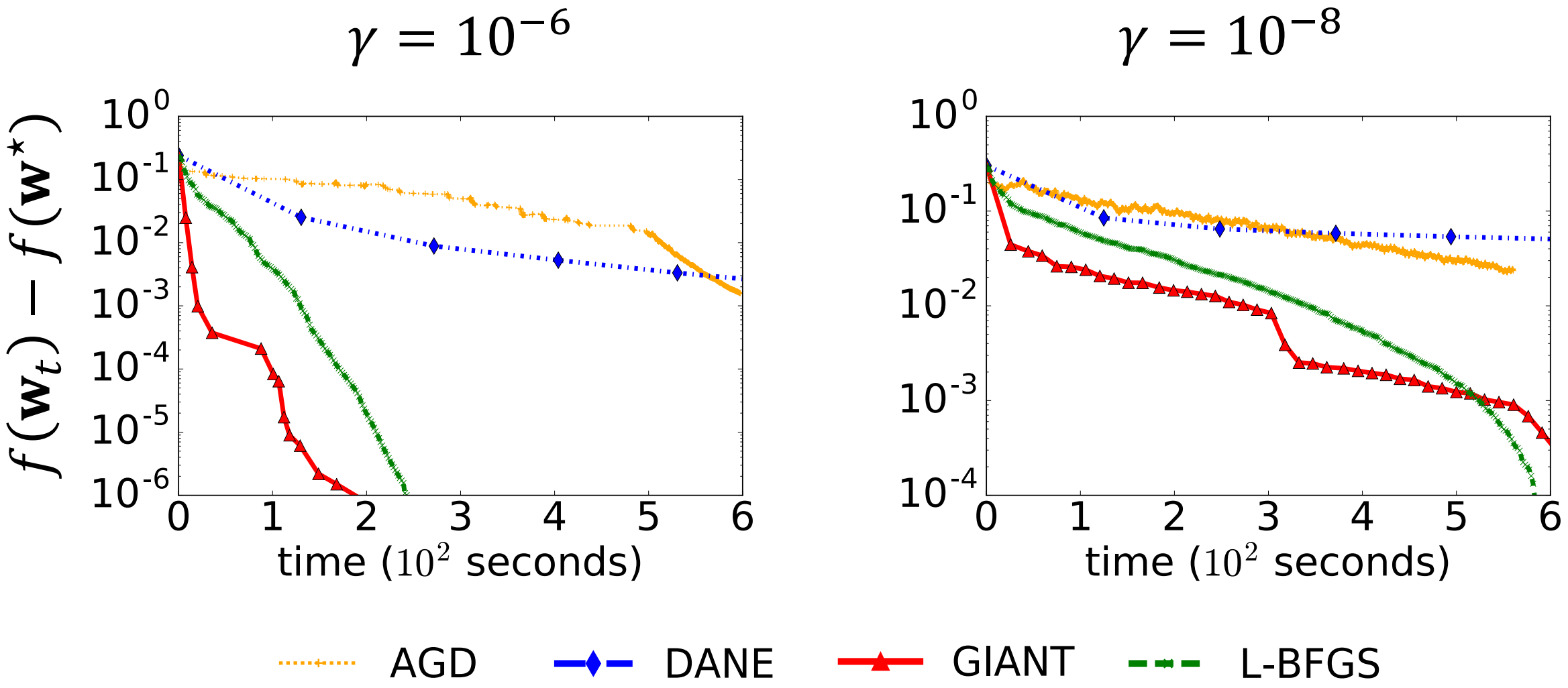
NERSC National Energy Research Scientific Computing Center

BERKELEY LAB
Bringing Science Solutions to the World

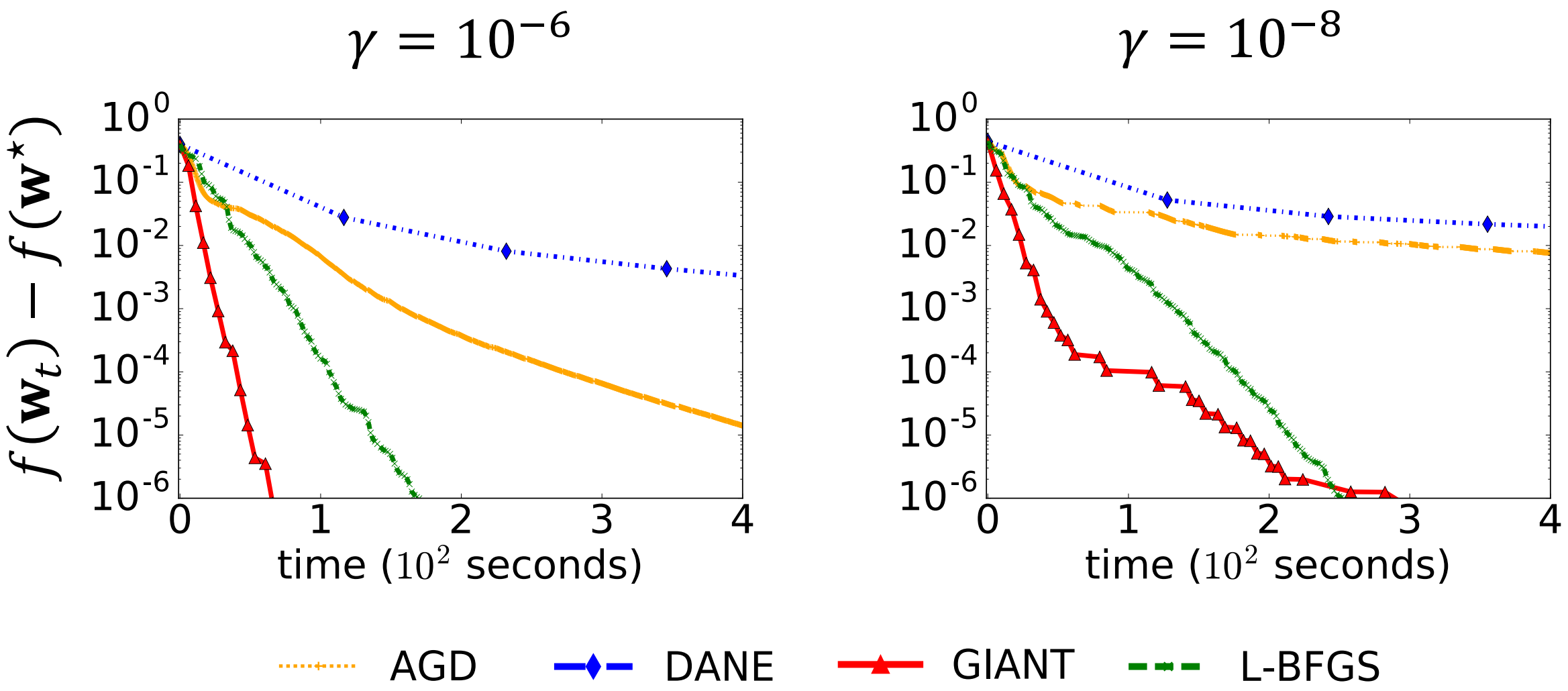
Experiment Environment

- `Spark 2.1.1 + Scala 2.11.8`
- `Cori Supercomputer (Cray XC40)`
  - 128 GB Memory / node
  - 32 Cores / node
- Use 15 nodes (480 CPU cores)

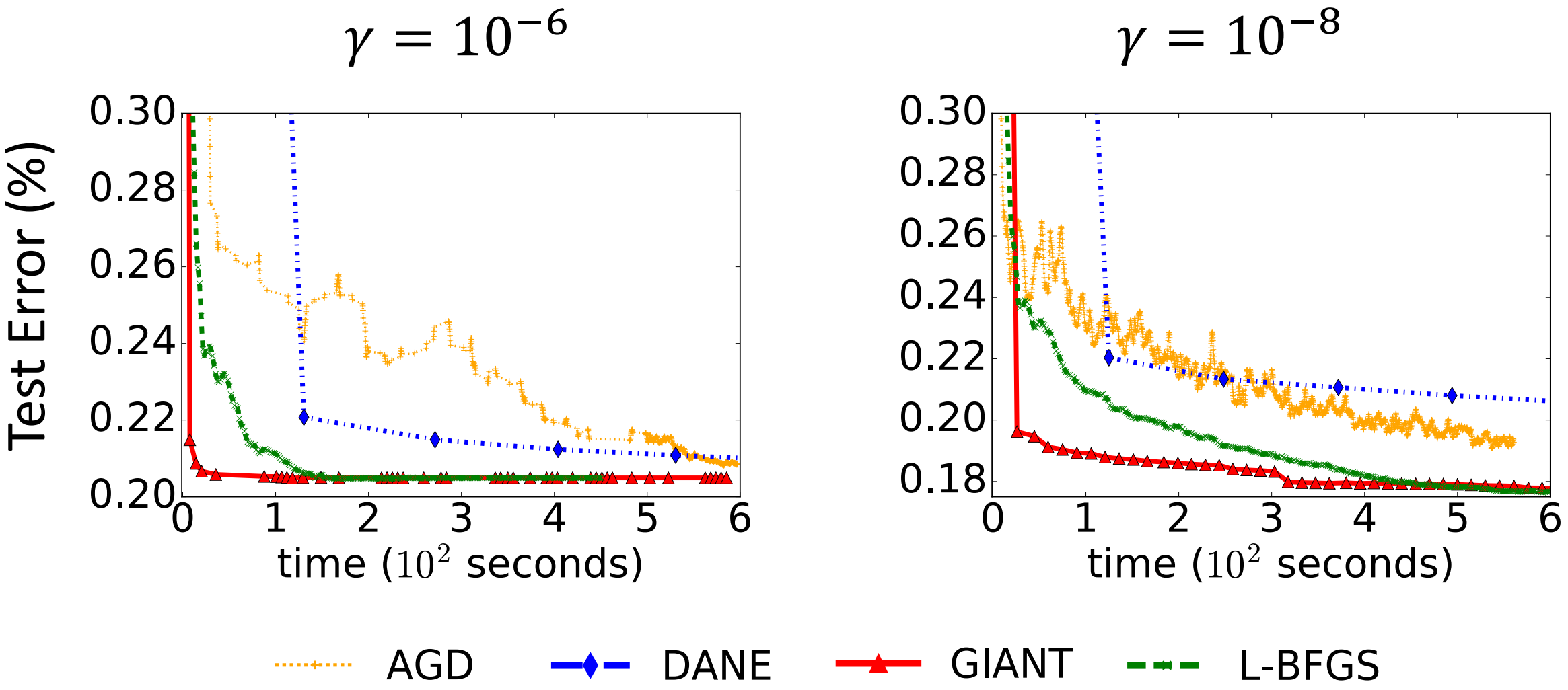
Covtype (n=581K, $\bar{d}$=10K), Training



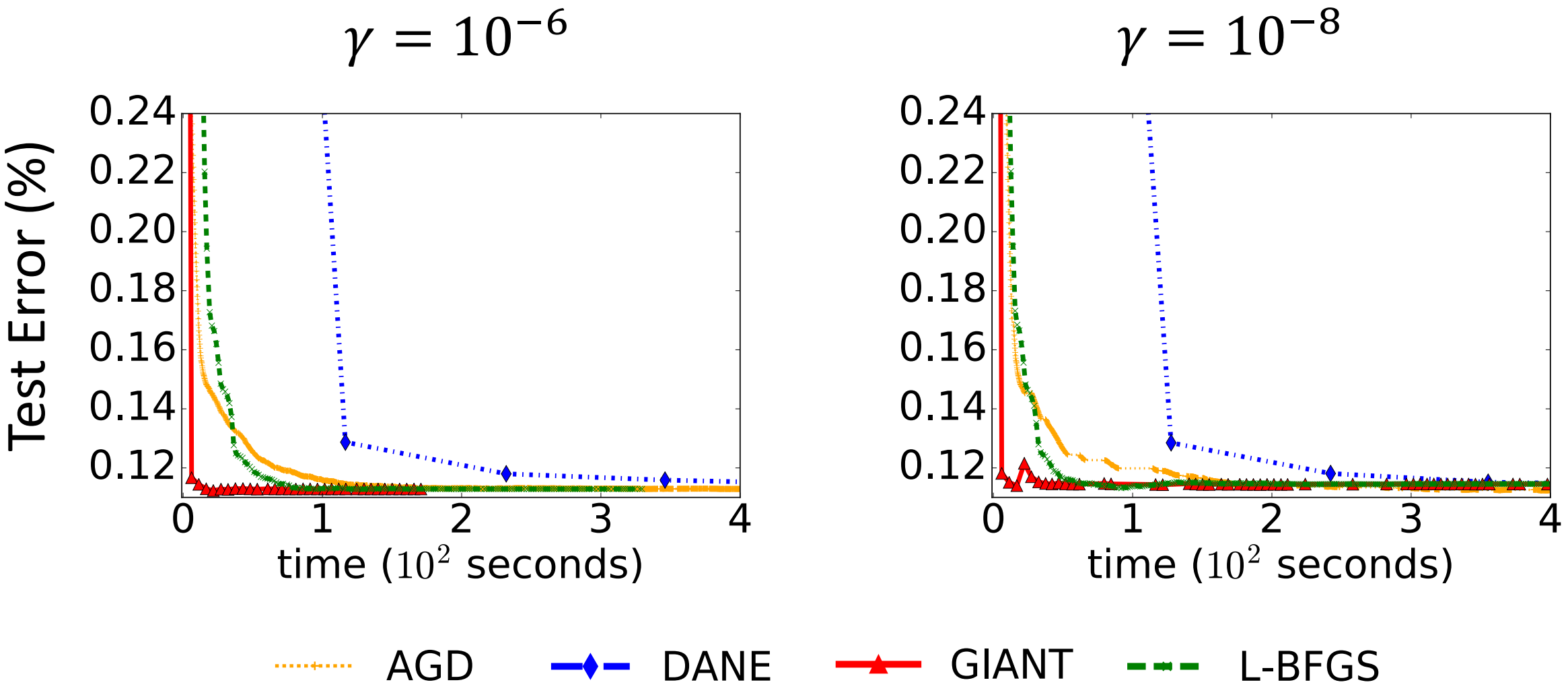
Epsilon (n=500K, $\bar{d}$=10K), Training



Covtype (n=581K, $\bar{d}$=10K), Test



Epsilon (n=500K, $\bar{d}$=10K), Test



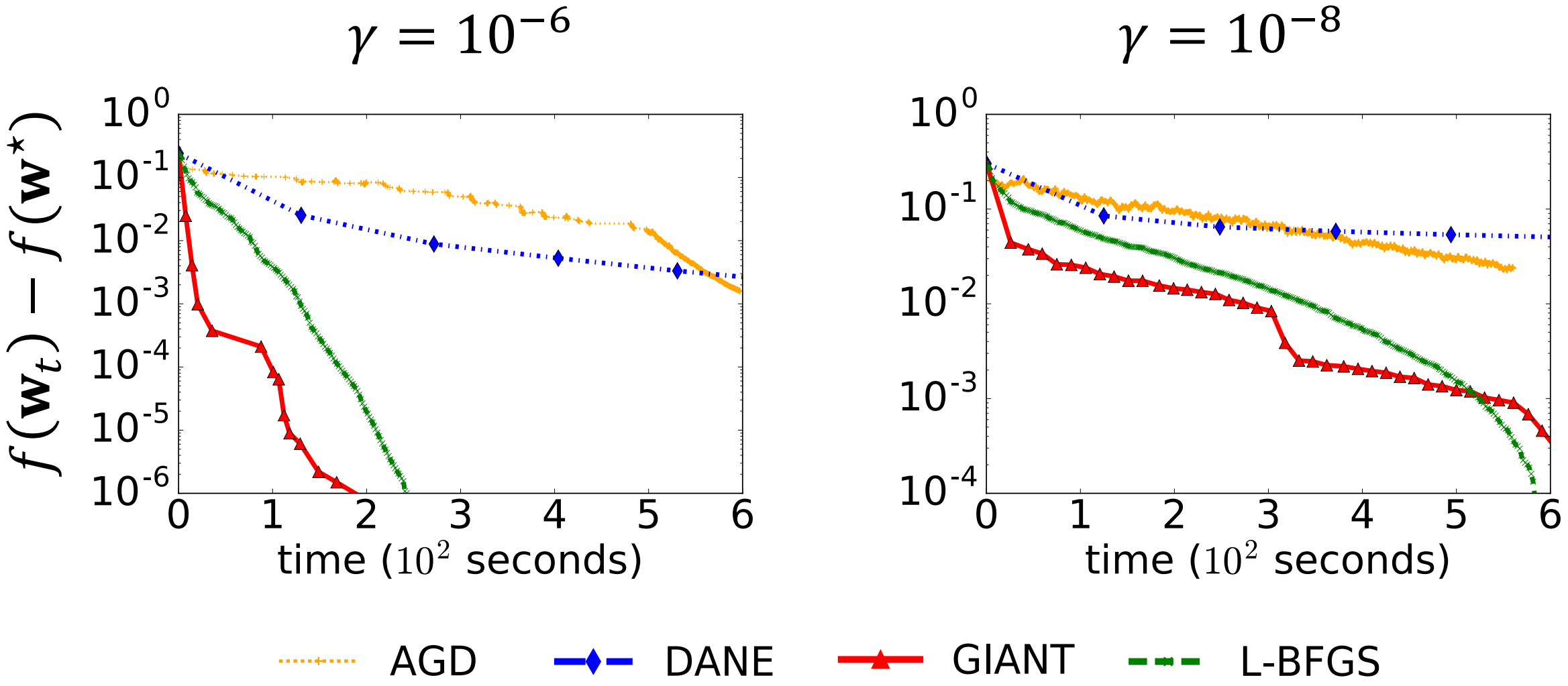
Scaling Experiments

- Make the Covtype data k times larger.
  1. Get k replicates of $\mathbf{X}$ and $\mathbf{y}$;
  2. Inject i.i.d. Gaussian noises to the $kn \times d$ feature matrix;
  3. Do random feature mapping to get 10K features.

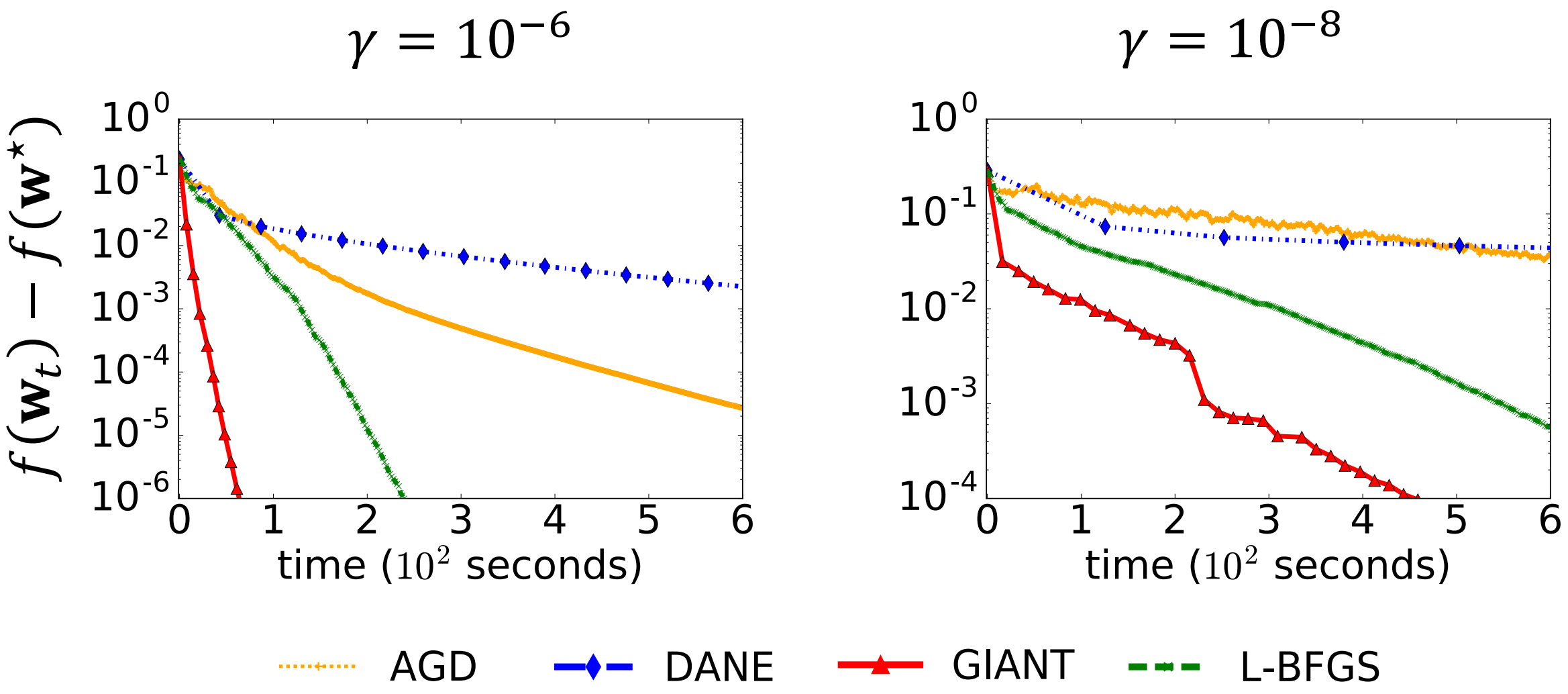
- Use k times more nodes.

- Set $k = 5$ and $k = 25$.

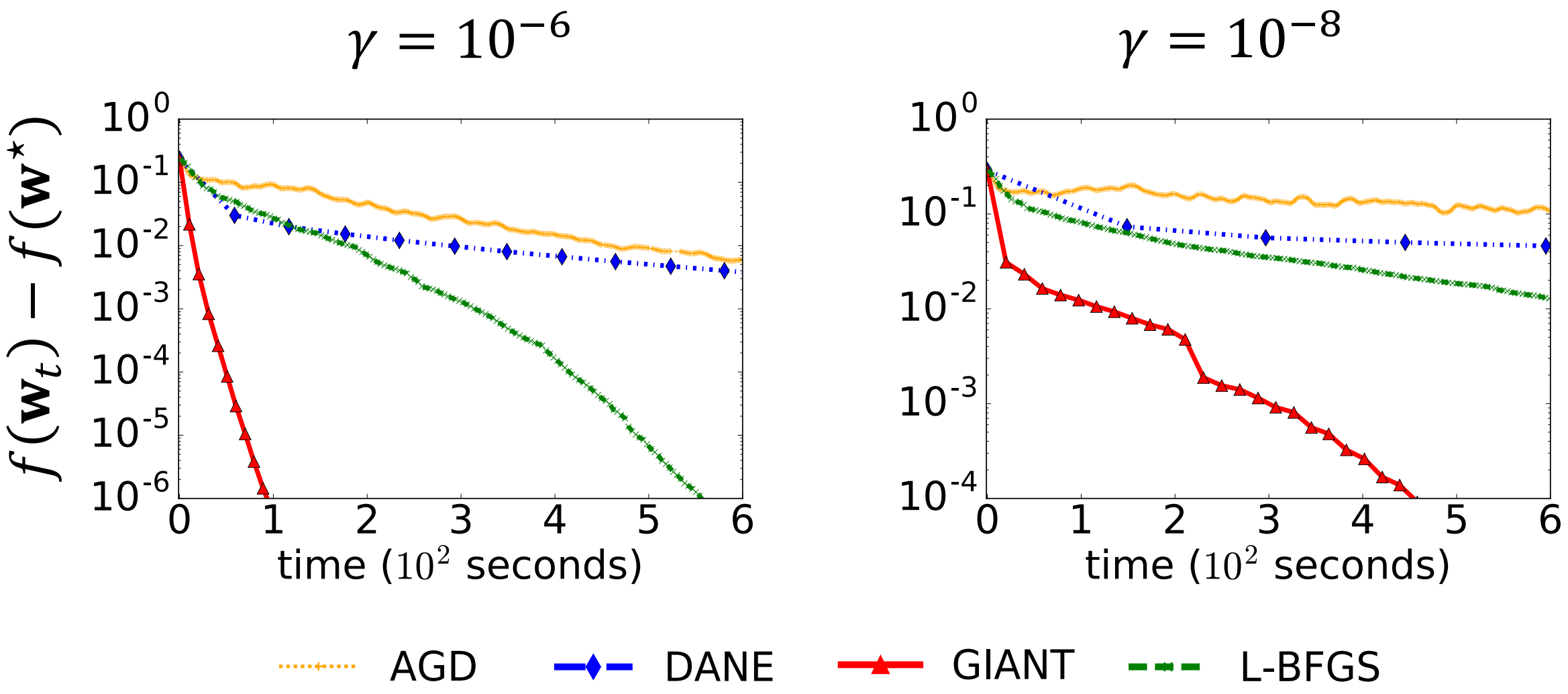
Original Data, 15 Nodes (480 Cores)



5x Larger Data, 75 Nodes (2.4K Cores)



25x Larger Data, 375 Nodes (12K Cores)

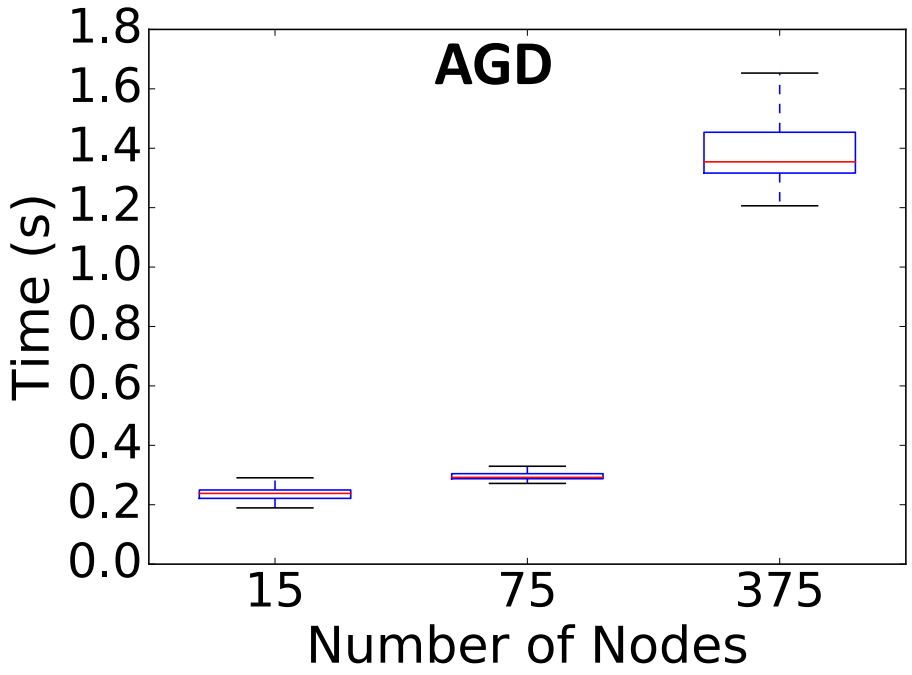


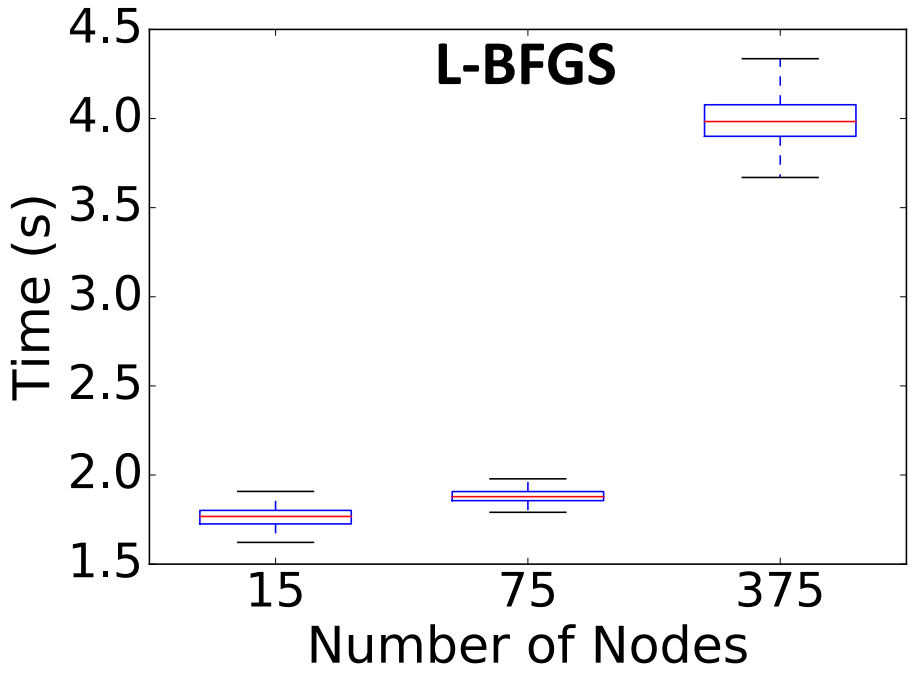
Why is GIANT More Scalable?

- As #Samples and #Nodes both increases by k times,
  - the **computational** costs remain **the same**;
  - the **communication** costs **increase**.

Why is GIANT More Scalable?

- As #Samples and #Nodes both increases by k times,
  - the computational costs remain the;
  - the communication costs increase.
- Per-iteration time of AGD and L-BFGS increases.





Why is GIANT More Scalable?

- As #Samples and #Nodes both increases by k times,
  - the computational costs remain the;
  - the communication costs increase.
- Per-iteration time of AGD and L-BFGS increases.
- Per-iteration time of GIANT marginally increases.
  - Because GIANT is computation-intensive.